移动平台开发书库



Android开发入门与实战体验

李佐彬　等编著



附赠超值 DVD 光盘

视频教学+电子课件+实例源代码

Android Practical Guide

剖析最热门的手机开发平台，传授最实用的应用开发技巧，

一本非常适于初学者学习手机编程的图书……





机械工业出版社
CHINA MACHINE PRESS

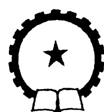
VISIT…

LANZAROTE Caliente.com

移动平台开发书库

Android 开发入门与实战体验

李佐彬　等编著



机 械 工 业 出 版 社

本书循序渐进地介绍了Android技术的基础知识，并通过实例教学的方式讲解了Android技术在各个领域的具体应用过程。全书分为16章，其中第1～5章是基础篇，讲解了Android的发展前景和开发环境的搭建过程；第6～13章是核心技术篇，详细讲解了Android技术的核心知识，并对程序优化进行了详细剖析；第14～16章是综合实战应用篇，通过3个综合实例详细讲解了Android技术常用的开发流程。

本书定位于Android的初、中级用户，可作为初学者的自学手册，也可以作为有一定基础的程序员的参考书。

图书在版编目（CIP）数据

Android开发入门与实战体验 / 李佐彬等编著．—北京：机械工业出版社，2011.7（2012.1重印）

（移动平台开发书库）

ISBN 978-7-111-34928-0

Ⅰ．①A…　Ⅱ．①李…　Ⅲ．①移动终端－应用程序－程序设计

Ⅳ．①TN929.53

中国版本图书馆CIP数据核字（2011）第103034号

机械工业出版社（北京市百万庄大街22号　邮政编码100037）

策划编辑：丁　诚

责任编辑：丁　诚

责任印制：杨　曦

保定市中画美凯印刷有限公司印刷

2012年1月•第1版第2次印刷

184mm×260mm•30.75印张•763千字

3501－5500册

标准书号：ISBN 978-7-111-34928-0

ISBN 978-7-89433-013-0（光盘）

定价：69.80元（含1CD）

凡购本书，如有缺页、倒页、脱页，由本社发行部调换

电话服务

社服务中心：（010）88361066

销 售 一 部：（010）68326294

销 售 二 部：（010）88379649

读者购书热线：（010）88379203

网络服务

门户网：http://www.cmpbook.com

教材网：http://www.cmpedu.com

封面无防伪标均为盗版

前　言

3G 时代的到来，使得更多内容丰富的应用程序布置在手机上成为可能，如视频通话、视频点播、移动互联网冲浪、在线看书 / 听歌、内容分享等。为了实现这些需求，需要有一个好的移动开发平台来支持。

Android 来历

2007 年 11 月推出的 Android 平台，是任何公司及个人都可以免费获取到源代码及 SDK 的一款手机开发平台。由于其开放性和优异性能，Android 平台得到了包括各大手机厂商和著名的移动运营商在内的业界广泛支持。

由于 Android 平台推出才 3 年，目前国内了解 Android 平台开发技术的程序员还不多，如何迅速地推广和普及 Android 平台软件开发技术，让越来越多的人参与到 Android 应用的开发中，是整个行业都在关注的一个话题。作者本人较早地从事了和 Android 相关的研究与开发工作，为了帮助读者更快地进入 Android 开发行列，精心编写了这本 Android 书。本书系统讲解了 Android 软件开发的基础知识，图文并茂地帮助读者学习和掌握 SDK、开发流程以及常用的 API 等。书中以讲述实战实例为导向，用一个个典型应用生动地引领读者进行项目开发实践。

巨大的优势

从技术角度而言，Android 与 iPhone 手机采用的 IOS 系统相似，采用 WebKit 浏览器引擎，具备触摸屏、高级图形显示和上网功能，用户能够在手机上查收电子邮件、搜索网址和观看视频节目。Android 手机比 iPhone 等其他手机更强调搜索功能，界面更强大，可以说是一种融入了全部 Web 应用的台。Android 的版本包括 Android 1.1、Android 1.5、Android 1.6、Android 2.0、Android 2.2、Android 2.3、Android 3.0 等。随着版本的更新，从最初的触屏到现在的多点触摸，从普通的联系人到现在的数据同步，从简单的 GoogleMap 到现在完善的导航系统，从基本的网页浏览到现在的 HTML5，Android 的功能越来越强大。此外，Android 平台不仅支持 Java、C、C++等主流的编程语言，还支持 Ruby、Python 等脚本语言，甚至 Google 专为 Android 开发的 Simple 语言，这使得 Android 开发拥有更自由的开发选择。

Android 作为新的平台、新的技术，为了帮助众多开发人员和爱好者进入 Android 开发领域，并提高程序开发水平，笔者编写了本书。

本书的内容

本书循序渐进地讲解了 Android 技术的基本知识，并通过实例的方式介绍了 Android 在各个领域的具体应用。本书内容新颖、知识全面、讲解详细，全书分为 16 章，其中第 1～7 章是基础篇，讲解了 Android 的发展前景和开发环境的搭建过程以及常用的核心知识；

第 8～16 章是典型应用篇，通过实例详细讲解了 Android 在现实中的常见领域的具体开发流程。

本书主要由李佐彬编写，参加编写的还有陈强、刘海洋、曹阳、李强、刁国庆、薛多鸯、张子言、李淑芳、陈德春、王梦、唐凯、王石磊、张家春、管西京、张玲玲等。

笔者本人毕竟水平有限，如有纰漏和不尽如人意之处在所难免，诚请读者提出意见或建议，以便修订并使之更臻完善。

编　者

目　　录

前言

第一篇　基　础　篇

第 1 章　揭开 Android 的神秘面纱 1
1.1　了解智能手机 1
1.1.1　智能手机的定义 1
1.1.2　智能手机的特点 1
1.1.3　主流智能手机操作系统 2
1.2　初识 Android 3
1.2.1　Android 手机 3
1.2.2　开放手机联盟 4
1.3　Android 特性 4
1.4　Android 组件结构应用程序框架 5
1.4.1　Android 组件结构 5
1.4.2　Android 应用程序框架 5
1.5　Android 何以脱颖而出 6
1.6　Android 模拟器 7
1.6.1　什么是模拟器 7
1.6.2　获取 Android 模拟器 8
第 2 章　搭建 Android 开发环境 9
2.1　开发 Android 应用前的准备 9
2.1.1　基本系统要求 9
2.1.2　Android 软件开发工具 9
2.2　全新的 Android 2.3 10
2.3　Windows 开发环境搭建 10
2.3.1　JDK、Eclipse、Android SDK 软件安装 10
2.3.2　设定 Android SDK Home 21
2.3.3　开发环境验证 21
2.3.4　创建 Android 虚拟设备(AVD) 22
2.4　常见的一些问题 24
第 3 章　Android SDK 简要介绍 29
3.1　Android SDK 基础 29
3.2　初步探寻 Android SDK 体系 29
3.2.1　Android SDK 目录结构 29
3.2.2　android.jar 及内部结构 30

3.2.3 SDK 文档及阅读技巧 ······ 31
3.2.4 SDK 工具集 ······ 32
3.3 解析 Android SDK 实例 ······ 33
第 4 章 看第一个应用程序 ······ 37
4.1 Hello World 应用程序分析 ······ 37
4.1.1 新建一个 Android 工程 ······ 37
4.1.2 设置工程的信息 ······ 37
4.1.3 编写代码和代码分析 ······ 38
4.1.4 运行项目 ······ 39
4.2 调试项目 ······ 40
4.2.1 设置断点 ······ 40
4.2.2 Debug 项目 ······ 41
4.2.3 断点调试 ······ 41
4.3 Dialog（对话框）简介 ······ 42
4.3.1 Android 对话框 ······ 42
4.3.2 Dialog 类详解 ······ 43
第 5 章 分解 Android 应用程序核心 ······ 46
5.1 Android 体系结构介绍 ······ 46
5.1.1 操作系统层 ······ 47
5.1.2 各种库和 Android 运行环境 ······ 47
5.1.3 应用程序 ······ 48
5.1.4 应用程序框架 ······ 48
5.2 Android 应用程序组成 ······ 48
5.2.1 Activity 介绍 ······ 48
5.2.2 Broadcast Intent Receiver 介绍 ······ 48
5.2.3 Service（服务）介绍 ······ 49
5.2.4 Content Provider 介绍 ······ 49
5.2.5 Intent 和 Intent Filter ······ 50
5.3 Android 应用工程文件组成 ······ 50
5.3.1 src 目录 ······ 51
5.3.2 AndroidManfest.xml 文件 ······ 52
5.3.3 常量的定义文件 ······ 53
5.4 应用程序的生命周期 ······ 54
5.5 Activity 的生命周期 ······ 55
5.5.1 Activity 的几种状态 ······ 55
5.5.2 分解剖析 Activity ······ 57
5.6 Android 进程与线程 ······ 57
5.6.1 进程 ······ 57
5.6.2 线程 ······ 58

5.6.3 线程安全的方法 58
第 6 章 Android 常用组件 59
6.1 UI（界面）组件 59
6.1.1 视图组件——View 59
6.1.2 视图容器组件——Viewgroup 59
6.1.3 布局组件——Layout 60
6.1.4 布局参数——LayoutParams 61
6.2 绚丽多彩——应用界面布局实例 62
6.2.1 编程实现 62
6.2.2 效果演示 69
6.3 Widget 组件详解 70
6.3.1 创建一个 Widget 组件 70
6.3.2 使用按钮 Button 71
6.3.3 使用文本框 TextView 72
6.3.4 使用编辑框 EditText 81
6.3.5 使用多项选择控件 CheckBox 82
6.3.6 使用单项选择控件 RadioGroup 85
6.3.7 使用下拉列表控件 Spinner 86
6.3.8 使用自动完成文本控件 AutoCompleteTextView 89
6.3.9 使用日期选择器控件 DatePicker 92
6.3.10 使用时间选择器 TimePicker 控件 94
6.3.11 联合应用 DatePicker 和 TimePicker 95
6.3.12 使用滚动视图控件 ScrollView 99
6.3.13 使用进度条控件 ProgressBar 99
6.3.14 使用拖动条控件 SeekBar 101
6.3.15 使用评分组件 RatingBar 102
6.3.16 使用图片视图控件 ImageView 104
6.3.17 使用图片按钮控件 ImageButton 108
6.3.18 使用切换图片控件 ImageSwitcher 和 Gallery 109
6.3.19 使用网格视图控件 GridView 112
6.3.20 使用标签控件 Tab 114
6.4 使用友好菜单控件 Menu 116
6.4.1 Menu 基础知识介绍 116
6.4.2 使用 Menu 实例 117
6.5 Intent 和 Activity 120
6.6 使用列表控件 ListView 124
6.6.1 ArrayAdapter 接受一个数组或者列表（List）作为参数来构建 124
6.6.2 使用 SimpleAdapter 125
6.7 使用互动对话框控件 Dialog 127

6.8 使用 Toast 和 Notification ······ 133
6.8.1 Toast 简介 ······ 134
6.8.2 Notification 简介 ······ 135
6.8.3 联合使用 Toast 和 Notification ······ 137
第 7 章 Android 组件高级应用 ······ 147
7.1 对话框中的进度条 ······ 147
7.2 再看布局组件 ······ 151
7.2.1 Android 的五大布局对象 ······ 152
7.2.2 垂直线性布局 ······ 156
7.2.3 水平线性布局 ······ 158
7.2.4 相对布局 ······ 160
7.2.5 表单布局 ······ 162
7.2.6 切换卡 ······ 165
7.3 联合使用 Spinner 和 setDropDownViewResource ······ 168
7.4 Gallery 和衍生 BaseAdapter 容器 ······ 172
7.5 文件搜索引擎 FileSearch ······ 176
7.6 用 AnalogClock 和 DigitalClock 控件实现模拟小时钟 ······ 180
7.7 使用 ListActivity（活动列表） ······ 186
7.7.1 ListActivity 介绍 ······ 186
7.7.2 ListActivity 使用实例 ······ 188
7.8 绘图处理 ······ 191
7.8.1 graphics 类基础 ······ 191
7.8.2 使用 Matrix 实现图片缩放 ······ 194
7.8.3 使用 Bitmap 和 Matrix 实现图片旋转 ······ 199
7.9 动态添加/删除 Spinner 菜单 ······ 204

第二篇 典型应用篇

第 8 章 Intent、Broadcast 和 Adapter 详解 ······ 210
8.1 Android 体系核心之“Intent” ······ 210
8.1.1 Intent 的构成 ······ 210
8.1.2 Intent 的解析 ······ 211
8.1.3 Intent 的基本用法 ······ 215
8.1.4 Intent 和 Activity ······ 220
8.1.5 Linkify 简介 ······ 222
8.2 Activity 的返回值和本地动作 ······ 223
8.2.1 Activity 返回值 ······ 223
8.2.2 Android 本地动作 ······ 225
8.2.3 Intent Filter 响应隐式 Intent ······ 226
8.2.4 Android 解析 Intent Filter ······ 228

8.2.5 隐式 Intent 响应与责任传递 228
8.3 用 Intent 来广播一个事件 229
8.3.1 广播事件 229
8.3.2 Broadcast Receiver 监听广播 230
8.3.3 Android 本地广播 232
8.4 Adapter 详解 232
8.4.1 Android 提供的 Adapter 233
8.4.2 用 Adapter 进行绑定 233
8.4.3 使用 ArrayAdapter 和 SimpleCursorAdapter 233
8.5 Internet 资源 235
8.5.1 连接 Internet 资源 236
8.5.2 利用 Internet 资源 236
第 9 章 Android 的数据存储操作 238
9.1 什么是 Android 数据存储 238
9.2 SharedPreferences 存储 238
9.2.1 SharedPreferences 存储类效率分析 238
9.2.2 应用实例 240
9.3 文件存储 243
9.4 再看 SQLite 存储方式 243
9.5 ContentProvider 存储 249
9.5.1 Content Provider 基础 250
9.5.2 使用 ContentProvider 251
9.6 网络存储 254
9.7 数据存储演练 255
9.7.1 SQLite 实现一个日记本 256
9.7.2 ContentProvider 操作一个数据日记本 264
第 10 章 电话与短信应用开发 278
10.1 IntentFilter 解析 278
10.1.1 动作测试 278
10.1.2 类别测试 279
10.1.3 数据测试 279
10.2 Intent 电话拨号处理 279
10.3 Intent 短信处理 285
10.3.1 创建 TinySMS 界面 286
10.3.2 设置权限 287
10.3.3 发送短信处理 287
10.4 Android 中电话和短信服务的包 288
第 11 章 Service 的全新服务 290
11.1 Service 深入剖析 290

11.1.1 Service 策略……290
11.1.2 创建 Service……290
11.1.3 使用 Service……291
11.1.4 与远程 Service 通信……292
11.1.5 设置权限……294
11.1.6 使用 Service 实例……294
11.2 Service 生命周期……300
11.3 提高 Service 优先级……300
11.4 Service 综合实例……301
第 12 章 使用位置服务和地图 API……305
12.1 位置服务……305
12.1.1 android.location 的功能类……305
12.1.2 Android 定位的基本流程……306
12.1.3 GPS 定位实例……309
12.1.4 构造 LocationProvider 查询条件……312
12.2 及时监听位置变化……313
12.2.1 Maps 库类介绍……313
12.2.2 LocationManager 及时监听为遏制位置信息……314
12.3 应用地图……315
12.3.1 使用前的设置……315
12.3.2 使用 Map API 密钥的基本流程……318
12.3.3 应用实例：使用 Map API 密钥……320
第 13 章 程序优化……327
13.1 实现高效 Java 编程规范的十条基础规则……327
13.2 命名规范……330
13.3 编写优秀代码的技巧……331
13.4 性能测试……335
13.4.1 背景……335
13.4.2 Java 代理……336
13.4.3 性能测试的步骤……337
13.4.4 计算机性能测试……338
13.4.5 内存性能测试……342
13.5 程序优化……345
13.5.1 Java 程序的基本优化……345
13.5.2 程序性能优化……354
13.5.3 何谓高效的 Android……359
13.5.4 Android 上的单元测试……365
13.6 UI 界面优化……368

第 14 章　网络 RSS 阅读器……376
14.1　基础知识介绍……376
14.1.1　RSS 的用途……376
14.1.2　RSS 阅读器……376
14.1.3　RSS 语法……377
14.2　SAX 技术介绍……378
14.2.1　SAX 的原理……378
14.2.2　基于对象和基于事件的接口……379
14.2.3　常用的接口和类……380
14.3　实现流程……383
14.4　具体实现……384
14.4.1　建立实体类……384
14.4.2　主程序文件 ActivityMain.java……387
14.4.3　实现 ContentHandler……390
14.4.4　主程序文件 ActivityShowDescription.java……393
14.4.5　主布局文件 main.xml……395
14.4.6　详情主布局文件 showdescription.xml……395
14.5　打包、签名和发布……398
14.5.1　申请会员……398
14.5.2　生成签名文件……400
14.5.3　使用签名文件……406
14.5.4　发布……408
第 15 章　通用手机助手……409
15.1　项目分析……409
15.1.1　构成模块……409
15.1.2　流程规划……410
15.2　具体实现……411
15.2.1　系统主界面……411
15.2.2　系统信息……414
15.2.3　硬件信息……424
15.2.4　软件信息……427
15.2.5　运行时信息……430
15.2.6　文件浏览器信息……435
15.2.7　AndroidManifest.xml 获取权限……438
第 16 章　Map 地图综合应用……439
16.1　项目分析……439
16.1.1　规划 UI 界面……440
16.1.2　数据存储设计……440
16.2　具体实现……441

16.2.1 新建工程……441
16.2.2 主界面……441
16.2.3 新建界面……444
16.2.4 设置界面……447
16.2.5 帮助界面……453
16.2.6 地图界面……456
16.2.7 数据存取……471
16.2.8 实现 Service 服务……476

第一篇　基　础　篇

第 1 章　揭开Android的神秘面纱

Android 是一种手机开发平台，它建立在 Java 基础之上，能够为手机软件开发提供快捷有效的解决方案。Android 功能十分强大，已经成为移动平台开发领域的新热点，在本章的内容中，将简单介绍 Android 的发展历程和背景，帮助读者初步了解 Android 的基本概况。

1.1　了解智能手机

为了更好地学习本书，让读者以更快的速度上手，本节将首先讲解和 Android 关系密切的智能手机的基本知识，为读者了解本章后面的内容打好基础。

1.1.1　智能手机的定义

所谓智能手机（Smartphone），是指“像个人电脑一样，具有独立的操作系统，可以由用户自行安装软件、游戏等第三方服务商提供的程序，通过此类程序来不断对手机的功能进行扩充，并可以通过移动通信网络来实现无线网络接入的这样一类手机的总称”。简单地说，智能手机就是一部像电脑一样可以通过安装软件来拓展手机基本功能的手机。

智能手机可以是传统的手机增加智能功能，比如 Symbian 操作系统的 S60 系列、Windows Mobile 操作系统的 Windows Mobile Smartphone 系列；可以是传统 PDA 加上手机通信功能，比如 Windows Mobile 操作系统的 Windows Mobile Pocket PCPhone 系列、Palm 操作系统的 Treo 系列；可以是其他独立类型，比如 Symbian 操作系统的 S80、UIQ，以及一些 Linux 操作系统的智能手机。然而，就近期的发展来看，这些智能手机的类型有相融合的趋势。智能手机有别于普通带触摸屏的手机。一般普通带触摸屏的手机使用的都是生产厂商自行开发的封闭式操作系统，所能实现的功能非常有限。

“智能手机（Smart Phone）”这个说法主要是针对“功能手机（Featurephone）”而言的，本身并不意味着这个手机有多“智能（Smart）”；从另一个角度来讲，所谓的“智能手机（SmartPhone）”就是一台可以随意安装和卸载应用软件的手机（就像电脑那样）。“功能手机（Featurephone）”是不能随意安装卸载软件的，Java 的出现使后来的“功能手机（Featurephone）”具备了安装 Java 应用程序的功能，但是 Java 程序的操作友好性，运行效率及对系统资源的操作都比“智能手机（SmartPhone）”差很多。

1.1.2　智能手机的特点

智能手机的主要特点如下。

- 具备普通手机的全部功能，能够进行正常的通话，发短信等手机应用。
- 具备无线接入互联网的能力，即需要支持 GSM 网络下的 GPRS 或者 CDMA 网络下的

CDMA 1X 或者 3G 网络。

- 具备 PDA 的功能，包括 PIM（个人信息管理），日程记事，任务安排，多媒体应用，浏览网页等功能。
- 具备一个具有开放性的操作系统，在这个操作系统平台上，可以安装更多的应用程序，从而使智能手机的功能可以得到无限地扩充。
- 具有人性化的一面，可以根据个人需要扩展机器的功能。
- 功能强大，扩展性能强，第三方软件支持多。

1.1.3 主流智能手机操作系统

当今比较著名的手机操作系统如下。

1．Symbian：Symbian OS（中文译音“塞班系统”）

Symbian 是由诺基亚、索尼爱立信、摩托罗拉、西门子等几家大型移动通信设备商共同出资组建的一个合资公司，专门研发手机操作系统，现已被诺基亚全额收购。Symbian 很像是 Windows 和 Linux 的结合体，有着良好的界面，采用内核与界面分离技术，对硬件的要求比较低，支持 C++，VB 和 J2ME。目前根据人机界面的不同，Symbian 体系的 UI（User Interface 用户界面）平台分为 Series 60、Series 80、Series 90、UIQ 等。Series 60 主要是给数字键盘手机用，Series 80 是为完整键盘所设计，Series 90 则是为触控方式而设计。

2．Windows Mobile

Windows Mobile 将熟悉的 Windows 桌面扩展到了个人设备中。Windows Mobile 是微软为手持设备推出的“移动版 Windows”，使用 Windows Mobile 操作系统的设备主要有 PPC 手机、PDA、随身音乐播放器等。Windows Mobile 操作系统有 3 种，分别是 Windows Mobile Standard、Windows Mobile Professional，Windows Mobile Classic。目前常用版本 Windows Mobile 6.5，最新的版本是 Windows Phone 7。目前生产 Windows Mobile 手机的主要厂商是 HTC，其他还有东芝，惠普，Mio，华硕，索爱，三星，LG，摩托罗拉，联想，斯达康，夏新、魅族等。

3．Linux

Linux 具有 2 个其他操作系统无法比拟的优势。第一，Linux 具有开放的源代码，能够大大降低成本；第二，既满足了手机制造商根据实际情况有针对性地开发自己的 Linux 手机操作系统的要求，又吸引了众多软件开发商对应用软件的开发，丰富了第三方应用。然而 Linux 操作系统有其先天的不足：入门难度高、熟悉其开发环境的工程师少、集成开发环境较差；由于微软 PC 操作系统源代码的不公开，基于 Linux 的产品与 PC 的连接性较差；尽管目前从事 Linux 操作系统开发的公司数量较多，但真正具有很强开发实力的公司却很少，而且这些公司之间开发是相互独立的，很难实现更大的技术突破。最初摩托罗拉公司非常推崇 Linux 平台，然而在和诺基亚的较量中不断失败，现在也不再那么热心 Linux 了，转而投向基于 Linux 的 Android 平台。

4．Palm

Palm 是流行的个人数字助理（PDA，又称掌上电脑）的传统名字，是一种手持设置形式，也被称做掌上电脑。广义上，Palm 是 PDA 的一种，由 Palm 公司发明，这种 PDA 上的操作系统也称为 Palm，有时又称为 Palm OS。狭义上，Palm 指 Palm 公司生产的 PDA 产品，以区别于 SONY 公司的 Clie 和 Handspring 公司的 Visor/Treo 等其他运行 Palm 操作系统的 PDA 产



品。其数据显示于一个液晶显示屏（LCD），显著特点之一是数据的基本输入方法：一个写入装置，叫做铁笔，能够单击显示器上的图标选择输入的项目。铁笔亦能用于手写到显示屏的表面输入包括文字和数字的信息（文字和数字），这被称为涂鸦。PalmPilot 系列产品原是由一家叫“PalmComputing”的公司所研发设计的，这个公司在历经两次并购后，成为 3Com 的一个事业部门，而后 Palm 公司又从 3Com 公司中独立出来，成为一个独立的公司。 北京时间 2009 年 2 月 11 日，Palm 公司 CEO Ed Colligan 宣布：以后将专注于 WebOS 和 Windows Mobile 的智能设备，而将不会再有基于“Palm OS”的智能设备推出，除了 Palm Centro 会在以后和其他运营商合作时继续推出。这对于 Palm 粉丝们来说，实在是一个令人扼腕叹息的消息，一个令人无奈却只能接受的消息。

5．BlackBerry

“黑莓”（BlackBerry）是加拿大 RIM 公司推出的一种移动电子邮件系统终端，其特色是支持推动式电子邮件、手提电话、文字短信、互联网传真、网页浏览及其他无线资讯服务。黑莓最强大也是最有优势的方面在于收发邮件，然而在中国，用手机收发邮件还不是很流行，所以黑莓在中国几乎是没有多大市场。

6. iPhone

iPhone 由苹果公司（Apple, Inc.）首席执行官史蒂夫·乔布斯在 2007 年 1 月 9 日举行的 Macworld 宣布推出，2007 年 6 月 29 日在美国上市，创新地将移动电话、可触摸宽屏 iPod 以及具有桌面级电子邮件、网页浏览、搜索和地图功能的突破性因特网通信设备这 3 种产品完美地融为一体。iPhone 引入了基于大型多触点显示屏和领先性新软件的全新用户界面，让用户用手指即可控制 iPhone。iPhone 还开创了移动设备软件尖端功能的新纪元，重新定义了移动电话的功能。有关 iPhone 的评价：“iPhone 是一款革命性的,不可思议的产品，比市场上的其他任何移动电话整整领先了五年，”苹果公司首席执行官史蒂夫·乔布斯说，“手指是我们与生俱来的终极定点设备，而 iPhone 利用它们创造了自鼠标以来最具创新意义的用户界面。”

7．Android

Android 一词的本义指“机器人”，是基于 Linux 平台的开源手机操作系统，该平台由操作系统、中间件、用户界面和应用软件组成，号称是首个为移动终端打造的真正开放和完整的移动软件。

1.2　初识Android

Android 平台采用了 WebKit 浏览器引擎，具备触摸屏、高级图形显示和上网功能，用户能够在手机上查看电子邮件、搜索网址和观看视频节目等，同时 Android 还具有比 iPhone 等其他平台更强的搜索功能，可以说是一种融入全部 Web 应用的平台。根据市场分析机构信息通信与媒体研究公司 Informa 的预测，Android 手机平台将在 3 年内超越苹果 iPhone 操作系统。我们完全相信未来 Android 的发展前景必将越来越广阔。

1.2.1　Android手机

2008 年 9 月 22 日，美国运营商 T-Mobile USA 在纽约正式发布了第一款 Android 手机——T-Mobile G1。该款手机为 HTC 公司代工制造，是世界上第一部使用 Android 操作系统的手

机，支持 WCDMA/HSPA 网络，理论下载速率为 7.2Mbit/s，并支持 Wi-Fi。

摩托罗拉的首款 Android 手机 CLIQ，如图 1-1 所示。搭载 Android 2.2 的 Moto ME722 如图 1-2 所示。



图 1-1　摩托罗拉的首款 Android



图 1-2　Android 2.2 的 Moto ME722

1.2.2　开放手机联盟

Android 系统由“开放手机联盟”运营开发，联盟成员包括摩托罗拉、HTC、PHILIPS、T-Mobile、高通、魅族、三星、LG 以及中国移动在内的 34 家企业。这些企业都将基于 Android 平台开发手机的新型业务，应用方面的通用性和互联性将在最大程度上得到保持。

这 34 家企业中并不包含把持 Symbian 的诺基亚，以及凭借着 iPhone 风光正在的苹果公司，美国运营商 AT&T 和 Verizon，当然微软也没有加入，独树一帜的加拿大 RIM 和他们的 Blackberry 也被挡在门外。

1.3　Android特性

Android 作为当今最火爆的智能手机操作系统之一，其主要特性如下。

- 应用程序框架支持组件的重用与替换。
- Dalvik 虚拟机针对移动设备做了优化。
- 内部集成浏览器，该浏览器基于开源的 WebKit 引擎。
- 支持 2D、3D 图形库，3D 图形库基于 OpenGL ES 1.0 （硬件加速可选）。
- #SQLite 用做结构化的数据存储。
- 支持多媒体，包括常见的音频、视频和静态印象文件格式（如 MPEG4、H.264、MP3、AAC、AMR、JPG、PNG、GIF）。
- 支持 GSM 电话 （依赖于硬件）。
- 支持蓝牙、EDGE、3G、Wi-Fi（依赖于硬件）。
- 支持照相机、GPS、指南针和加速度计（依赖于硬件）。
- 拥有丰富的开发环境，包括设备模拟器、调试工具、内存及性能分析图表和 Eclipse 集成开发环境插件。



1.4　Android组件结构应用程序框架

和主流的开发工具一样，Android 也有自己的组件，本节将简要介绍 Android 组件结构应用程序框架的基本知识。

1.4.1　Android组件结构

Android 采用了软件堆层（Software Stack，又名软件叠层）的架构，主要分为 3 部分：低层以 Linux 核心为基础，只提供基本功能，其他的应用软件则由各公司自行开发，以 Java 作为编写程序的一部分。

1.4.2　Android应用程序框架

Android 是同一个核心应用程序包一起发布的，该应用程序包包括 E-mail 客户端、SMS 短消息程序、日历、地图、浏览器、联系人管理程序等，这些应用程序都是用 Java 编写的。

开发者也完全可以访问核心应用程序所使用的 API 框架，该应用程序架构用来简化组件软件的重用；任何一个应用程序都可以发布它的功能块并且任何其他的应用程序都可以使用其所发布的功能块(不过得遵循框架的安全性限制)。该应用程序重用机制使得组件可以被用户替换。

以下所有的应用程序都由一系列的服务和系统组成，主要包括如下内容。

- 一个可扩展的视图（View）：可以用来建应用程序，包括列表（Lists），网格（Grid），文本框（Text Boxes），按钮（Button），甚至包括一个可嵌入的 Web 浏览器。
- 内容管理器（Content Providers）：使得应用程序可以访问另一个应用程序的数据（如联系人数据库），或者共享它们自己的数据。
- 一个资源管理器（Resource Manager）：提供非代码资源的访问，如本地字符串，图形，和分层文件（Layout Files）。
- 一个通知管理器(Notification Manager)：使得应用程序可以在状态栏中显示客户通知信息。
- 一个活动类管理器（Activity Manager）：用来管理应用程序生命周期并提供常用的导航回退功能。

一个 Android 程序编译运行后的效果如图 1-3 所示。

Android 系统提供给应用开发者的本身就是一个框架，所有的应用开发都必须遵守这个框架的原则。在开发应用时就是在这个框架上进行扩展，下面来看看 Android 这个框架都有些什么功能可供大家使用。

- Android.app：提供高层的程序模型和基本的运行环境。
- Android.content：包含对各种设备上的数据进行访问和发布。
- Android.database：通过内容提供者浏览和操作数据库。
- Android.graphics：底层的图形库，包含画布、颜色过滤、点、矩形，可以将它们直接绘制到屏幕上。

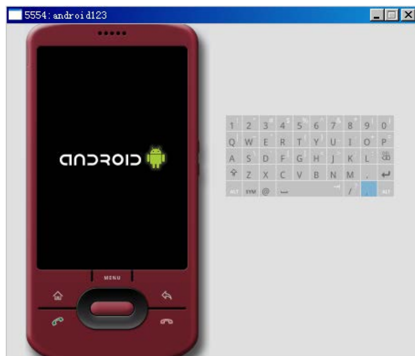


图 1-3　Android 程序运行效果

- Android.location：定位和相关服务的类。
- Android.media：提供一些类管理多种音频、视频的媒体接口。
- Android.net：提供帮助网络访问的类，超过通常的 java.net.* 接口。
- Android.os：提供了系统服务、消息传输和 IPC 机制。
- Android.opengl：提供 OpenGL 的工具。
- Android.provider：提供访问 Android 内容提供者的类。
- Android.telephony：提供与拨打电话相关的 API 交互。
- Android.view：提供基础的用户界面接口框架。
- Android.util：涉及工具性的方法，例如时间日期的操作。
- Android.webkit：默认浏览器操作接口。
- Android.widget：包含各种 UI 元素（大部分是可见的）在应用程序的布局中使用。

1.5 Android何以脱颖而出

智能手机追求智能和速度，新兴的 Android 系统之所以能在激烈的竞争中脱颖而出，得益于它拥有的无可比拟的性能优点。

1．Android技术易学易用

Android 是通过 Java 语言开发的，只要读者具备 Java 开发基础，就能很快地上手。即使没有编程经验的门外汉，也可以在突击学习 Java 基础之后快速掌握 Android。另外，由于 Android 完全支持 2D、3D 和数据库技术，并且和浏览器实现了集成。所以通过 Android 平台，程序员可以迅速、高效地开发出绚丽多彩的应用。

2．通过Android Market可轻松获益

与苹果公司的 APP Store 一样，Android 也拥有自己的软件商店 Android Market。开发人员可将应用程序在 Android Market 上发布，Android 用户即可随意下载自己喜欢的程序。Android Market 将是一个很大的软件市场，开发者可以发布并销售自己的软件。只要开发的软件程序足够吸引人，就可以获得很好的经济收益，从而达到学习、赚钱两不误。Android Market 地址是http://www.Android.com/market/，界面如图 1-4 所示。



图 1-4　Android Market 主界面



1.6　Android模拟器

我们都知道程序开发需要调试，只有经过调试之后才能知道程序是否能正确运行。作为一款手机系统，我们怎么样才能在电脑平台之上调试 Android 程序呢？不用担心，其官方为我们提供了模拟器来解决这个问题。

1.6.1　什么是模拟器

我们开发 Android 程序之后，需要使用 Android 手机环境来运行，以验证我们开发程序的效果。但是如果用真机来调试程序，会影响开发效率。为了解决这个问题，Android 为我们推出了模拟器。所谓模拟器，就是指在电脑上模拟 Android 系统，可以用这个模拟器来调试并运行开发的 Android 程序。开发人员不需要真实的 Android 手机，通过电脑即可模拟运行一部手机，并开发出手机应用程序。模拟器在电脑上模拟运行的效果如图 1-5 所示。



图 1-5　模拟器模拟手机效果

模拟器给 Android 程序开发者在开发和测试上带来了很大的便利。无论在 Windows 下还是在 Linux 下，Android 模拟器都可以顺利运行，并且官方提供了 Eclipse 插件，可将模拟器集成到 Eclipse 的 IDE 环境。当然，也可以从命令行启动 Android 模拟器。

当然 Android 模拟器不能完全替代真机，与真机的差异如下。

- 模拟器不支持呼叫和接听实际来电。
- 模拟器不支持 USB 连接。
- 模拟器不支持相机/视频捕捉。
- 模拟器不支持音频输入（捕捉）。
- 模拟器不支持扩展耳机。
- 模拟器不能确定连接状态。
- 模拟器不能确定电池电量水平和交流充电状态。
- 模拟器不能确定 SD 卡的插入/弹出。
- 模拟器不支持蓝牙。

1.6.2 获取Android模拟器

获取模拟器的方法非常简单，我们既可以从官方网站（http://developer.Android.com/）免费下载单独的模拟器，也可以从网络中用搜索关键字的方法获取。网络中推出了很多版本的 Android 模拟器，例如，有专门针对游戏玩家的，有专门针对不同手机型号的。当下载 Android SDK 后，解压后在其 SDK 的根目录下有一个名为“tools”的文件夹，此文件夹下包含了完整的模拟器和一些非常有用的工具。

Android SDK 中包含的模拟器的功能非常齐全，电话本、通话等功能都可正常使用（当然没办法真的从这里打电话）。甚至其内置的浏览器和 Maps 都可以联网。用户可以使用键盘输入，鼠标单击模拟器按键输入，如图 1-6 所示。甚至还可以使用鼠标单击、拖动屏幕进行操纵。

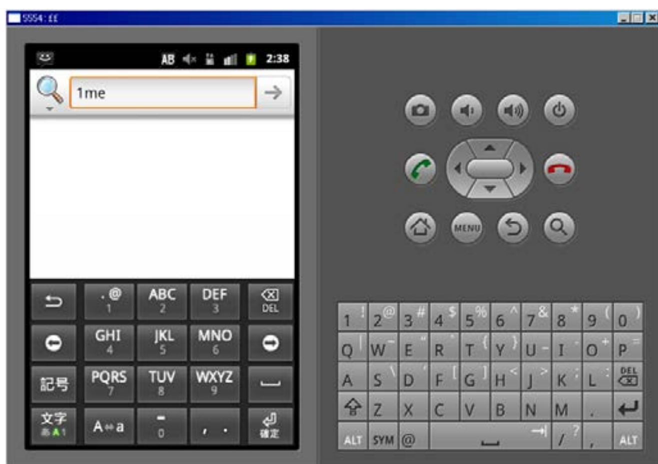


图 1-6 在模拟器中输入文字

第 2 章　搭建Android开发环境

“工欲善其事，必先利其器”，要想高效地完成一件工作，必须有一个合适的工具。对于编程人员来说，开发工具同样至关重要。选择一个高效的开发工具后，开始搭建一个科学的开发平台是开发人员的首要任务。本章将详细介绍搭建 Android 开发环境的基本知识。

2.1　开发Android应用前的准备

在搭建开发环境前，需要了解安装开发工具所需要的硬件和软件配置条件。下面将讲解搭建 Android 开发环境的准备工作。

2.1.1　基本系统要求

开发基于 Android 的应用软件所需要的开发环境如表 2-1 所示。

表 2-1　开发系统所需求参数

项目	版 本 要 求	说 明	备 注
操作系统	Windows XP 以上版本、Mac OS、Linux	根据自己的电脑自行选择	选择自己最熟悉的操作系统
软件开发包	Android SDK	选择最新版本的 SDK	
IDE	Eclipse IDE+ADT	Eclipse3.3 (Europa)、3.4 (Ganymede)和 ADT(Android Development Tools)开发插件	选择“for Java Developer”
其他	JDK Apache Ant	Java SE Development Kit 5 或 6 Linux 和 Mac 上使用 Apache Ant 1.6.5+、Windows 上使用 1.7+版本	（仅有 JRE 是不够的，必须要有 JDK），不兼容 Gnu Java 编译器(gcj)

2.1.2　Android软件开发工具

Android 软件开发需要以下主要工具。

- JDK：可以到 http://java.sun.com/javase/downloads/index.jsp 下载。
- Eclipse（Europa）：可以到 http://www.eclipse.org/downloads/下载 Eclipse IDE for Java Developers。
- Android SDK：可以到 http://developer.android.com 下载。
- 配套的开发插件。

注意：Eclipse 可以到对应的网站下载安装，如果通过网络远程安装不成功，可以下载到本地安装。

2.2　全新的Android 2.3

北京时间 2010 年 12 月 7 日，Google 公司正式发布了最新的智能手机操作系统 Android 2.3。虽然在版本号方面 Android 2.3 相对于 Android 2.2 而言的提升并不算多，但是从功能提升以及界面改进上来看，变化还是十分明显的。

（1）新的 UI 界面

Android 2.3 在配色方面更多的使用了绿色，例如状态栏、通话图标以及其他一些功能图标。全新的系统使界面的反应速度有了一定程度的提升，用户使用 Android 系统时将会觉得使用起来更为流畅。

（2）革新的输入方式

使用过 Android 原生输入法的用户一定会抱怨他们的输入法在输入时存在一定的延迟，所以无法立即看见自己输入的内容。但是这样的现象在 Android 2.3 上发生了根本的改进，用户将会在屏幕上立即看见自己输入的字母，而不会出现延迟的现象。

Android 2.3 还加入了全新的拼写检查机制，只要用户在文本中选中一段字母，Android 2.3 就能够自动地给出一些建议的拼写答案，而用户需要做的就是从中选择正确的结果。

不过，在新的输入法中最为令人兴奋的还是加入了对于虚拟组合键的支持，例如我们现在可以在英文输入模式下使用〈Shift〉键来输入数字，而不需要我们切换到数字输入的模式。

（3）更丰富的游戏体验

为了提升 Android 系统在游戏方面的表现，Google 公司在 Android 2.3 中加入了更多的程序接口以方便程序员的使用，同时在传感器的支持方面也有了一定的提升，例如 Android 2.3 已经加入了对于陀螺仪的支持，因此在未来的 Android 游戏中我们将会拥有更为丰富的用户体验。

（4）全新的交流方式

Android 2.3 首次加入了对于 VOIP 以及 SIP 的支持，这对于目前发展迅猛的互联网通话是有很大帮助的。Android 2.3 还加入了对于近距离支付也就是 NFC 功能的支持。而在前置摄像头方面虽然还没有视频通话的支持，但是 Google 公司已经允许用户使用第三方应用程序来调用前置的摄像头了，这也预示着未来手机上视频聊天软件的增多。

2.3　Windows开发环境搭建

因为当前主流的操作系统是 Windows，所以在此首先讲解在 Windows 中搭建 Android 开发环境的基本知识。

2.3.1　JDK、Eclipse、Android SDK软件安装

本书所讲的安装是以 Windows XP SP2 为平台的，安装的软件为 JDK 1.6、Eclipse 3.3、ADT2.3、Android SDK。下面具体介绍各自的安装步骤，并且在配套的视频中有详细的介绍。

1．安装 JDK

第 1 步：安装 Eclipse 的开发环境需要 JRE 的支持，在 Windows 上安装 JRE/JDK 非常简



单，首先在 SUN 官方网站下载，网址为 http://developers.sun.com/downloads/，如图 2-1 所示。

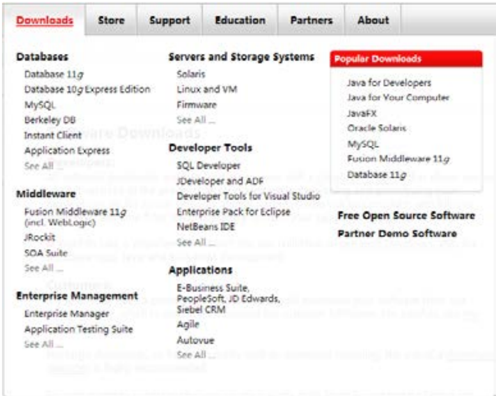


图 2-1　SUN 官方下载页面

第 2 步：在图 2-1 中可以看到有很多版本，运行 Eclipse 时虽然只需要 JRE 就可以了，但是在开发 Andriod 应用程序的时候，需要完整的 JDK（JDK 已经包含了 JRE），且要求其版本在 1.5 以上，这里选择 Java SE (JDK) 6，其下载页面如图 2-2 所示。

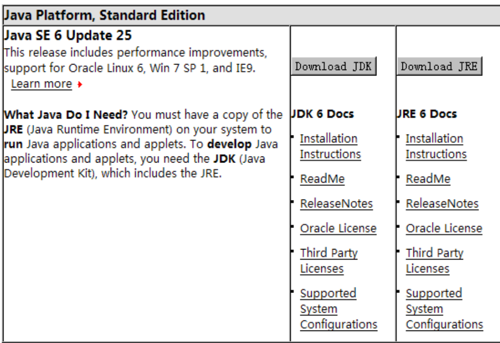


图 2-2　JDK 下载页面

第 3 步：在图 2-2 中选择“Java SE 6 Update 25 (JDK or JRE)”。单击其右侧的“Download”按钮，出现让用户选择其操作系统和语言的界面，在此首先选择“Windows x86”，然后单击“Download”按钮，如图 2-3 所示。

Java SE Development Kit 6 Update 25

Product / File Description	File Size	Download
Linux x86 - RPM Installer	76.85 MB	jdk-6u25-linux-i586-rpm.bin
Linux x86 - Self Extracting Installer	81.11 MB	jdk-6u25-linux-i586.bin
Linux x64 - RPM Installer	77.06 MB	jdk-6u25-linux-x64-rpm.bin
Linux x64 - Self Extracting Installer	81.36 MB	jdk-6u25-linux-x64.bin
Solaris x86 - Self Extracting Binary	81.00 MB	jdk-6u25-solaris-i586.sh
Solaris x86 - Packages - tar.Z	136.67 MB	jdk-6u25-solaris-i586.tar.Z
Solaris SPARC - Self Extracting Binary	85.96 MB	jdk-6u25-solaris-sparc.sh
Solaris SPARC - Packages - tar.Z	141.11 MB	jdk-6u25-solaris-sparc.tar.Z
Solaris SPARC 64-bit - Self Extracting Binary	12.24 MB	jdk-6u25-solaris-sparcv9.sh
Solaris SPARC 64-bit - Packages - tar.Z	15.58 MB	jdk-6u25-solaris-sparcv9.tar.Z
Solaris x64 - Self Extracting Binary	8.49 MB	jdk-6u25-solaris-x64.sh
Solaris x64 - Packages - tar.Z	12.25 MB	jdk-6u25-solaris-x64.tar.Z
Windows x86	76.66 MB	jdk-6u25-windows-i586.exe
Windows x64	67.27 MB	jdk-6u25-windows-x64.exe

图 2-3　选择“Windows x86”

第 4 步：经过上述操作后，就已经开始下载安装文件“jdk-6u25-windows-i586.exe”。

注意：在此需要会员用户登录后才能下载。

第 5 步：下载完成后双击“jdk-6u25-windows-i586.exe”文件开始进行安装。首先将弹出“许可证协议”对话框，在此单击“接受”按钮，如图 2-4 所示。



图 2-4 “许可证协议”对话框

第 6 步：弹出“安装路径”对话框，在此选择文件的安装路径，如图 2-5 所示。

第 7 步：单击“下一步”按钮，开始进行安装，如图 2-6 所示。



图 2-5 “安装路径”对话框



图 2-6 开始安装

第 8 步：完成后弹出“目标文件夹”对话框，在此选择目标文件夹的路径，如图 2-7 所示。



图 2-7 “目标文件夹”对话框



第 9 步：单击“下一步”按钮后，继续安装，如图 2-8 所示。



图 2-8　继续安装

第 10 步：完成后弹出“完成”对话框，单击“完成”按钮，完成整个安装过程，如图 2-9 所示。



图 2-9　完成安装

完成安装后，我们可以检测是否安装成功，具体的方法是：依次单击“开始”→“运行”，在运行框中输入“cmd”并按〈Enter〉键，在打开的 CMD 窗口中输入 java –version，如果显示如图 2-10 所示的提示信息，则说明安装成功。

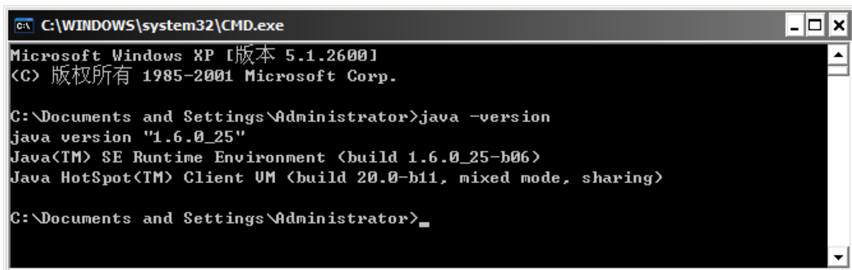


图 2-10　CMD 窗口

如果检测没有安装成功，则需要将它的目录的绝对路径添加到系统的路径中，具体做法如下。

第 1 步：右键依次单击“我的电脑”→“属性”→“高级”，单击下面的“环境变量”，在下面的“系统变量”处选择新建在变量名处输入“JAVA_HOME”，变量值中输入刚才的目录，比如“C:\Program Files\Java\jdk1.6.0_25”，如图 2-11 所示。

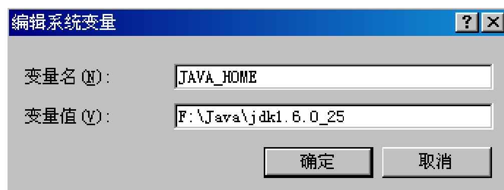


图 2-11　设置系统变量

第 2 步：再新建一个变量名为 classpath，变量值为“.;%JAVA_HOME%/lib/rt.jar;%JAVA_HOME%/lib/tools.jar”，确定后找到路径的变量，双击或单击编辑，在变量值最前面加上“%JAVA_HOME%/bin;”，如图 2-12 所示。

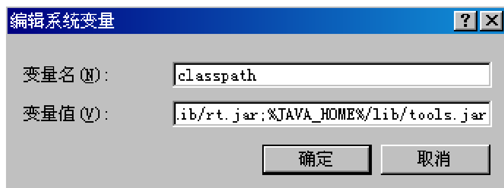


图 2-12　设置系统变量

第 3 步：再次依次单击“开始”→“运行”，在运行框中输入“cmd”并按〈Enter〉键，在打开的 CMD 窗口中输入 java –version，如果显示如图 2-13 所示的提示信息，则说明安装成功。

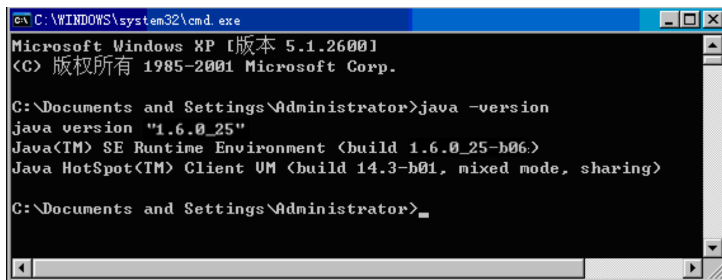


图 2-13　CMD 界面

注意：上述变量设置中，是按照笔者本人的安装路径设置的，笔者安装 JDK 的路径是 C:\Program Files\Java\jdk1.6.0_25。

2．安装 Eclipse

在安装好 JDK 后，就可以接着安装 Eclipse 了，具体过程如下。

第 1 步：打开 Eclipse 的下载页面 http://www.eclipse.org/downloads/，如图 2-14 所示。





图 2-14　下载页面

第 2 步：在图 2-14 中选择“Eclipse IDE for Java Developers (92 MB)”，在其下载的镜像页面，选择离用户最近的镜像即可（一般推荐的下载速度就不错），如图 2-15 所示。

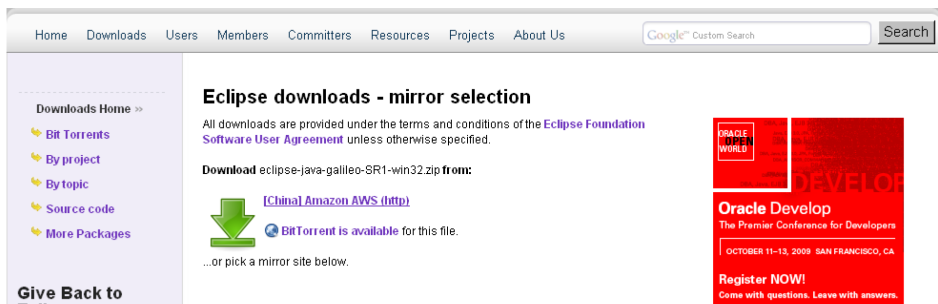


图 2-15　选择镜像

第 3 步：下载完成后，找到下载的压缩包“eclipse-java-galileo-SR1-win32.zip”。Eclipse 无须执行安装程序，解压此压缩文件就可以用，不过一定要先安装 JDK。在此假设 Eclipse 解压后存放的目录为“F:\eclipse”。

第 4 步：进入解压后的目录，此时可以看到一个名为“eclipse.exe”的可执行文件，双击此文件直接运行，Eclipse 能自动找到用户先期安装的 JDK 路径，启动界面如图 2-16 所示。

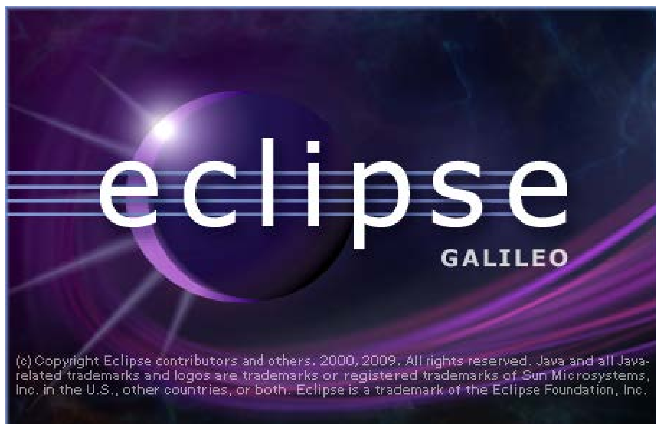


图 2-16　eclipse 启动界面

第 5 步：因为是第一次安装、启动 Eclipse，将会看到选择工作空间的提示，如图 2-17 所示。

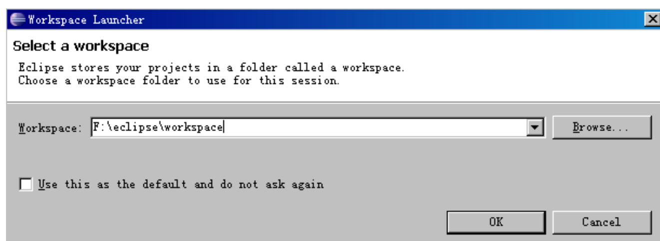


图 2-17　选择工作空间

此时单击“OK”按钮后，完成 Eclipse 的安装。系统进入初始欢迎界面，如图 2-18 所示。

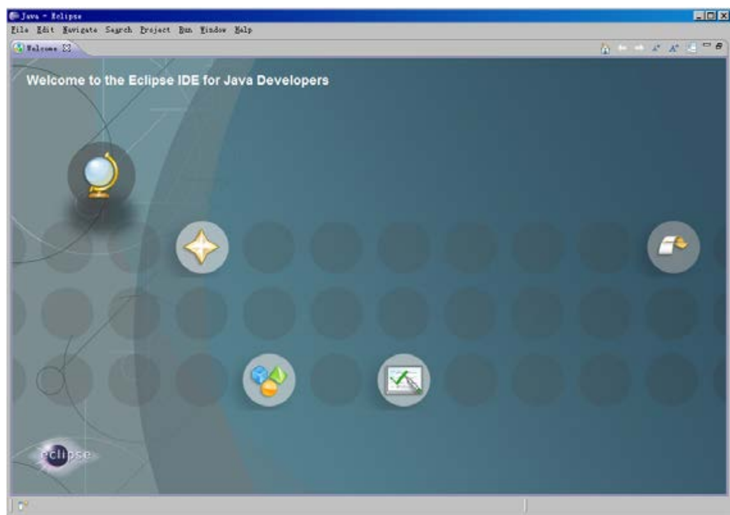


图 2-18　初始欢迎界面

3. 安装 Android SDK

安装 JDK 和 Eclipse 完毕后，接下来需要下载安装 Andriod 的 SDK，具体流程如下。

第 1 步：打开 Android 开发者社区网址 http://developer.android.com/，然后转到 SDK 下载页面下载“android-sdk_r06-windows.zip”，如图 2-19 所示。

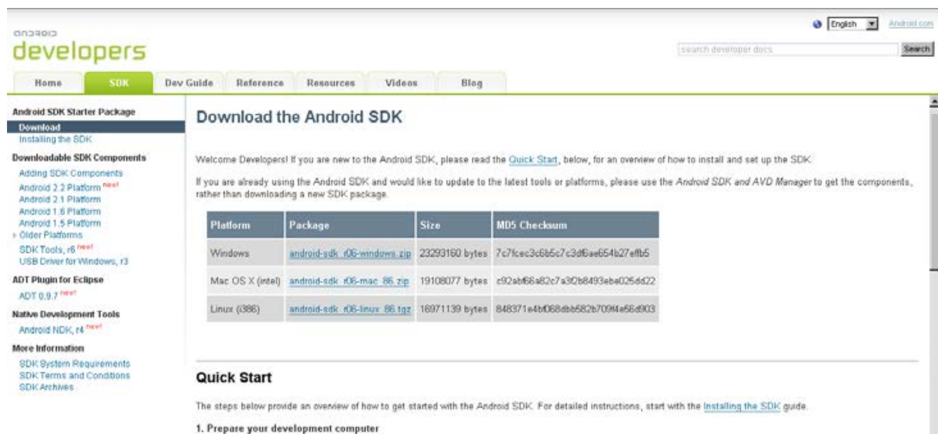


图 2-19　SDK 下载页面



第 2 步：选择用于 Windows 平台的“android-sdk_r06-windows.zip”，打开下载页面，如图 2-20 所示。

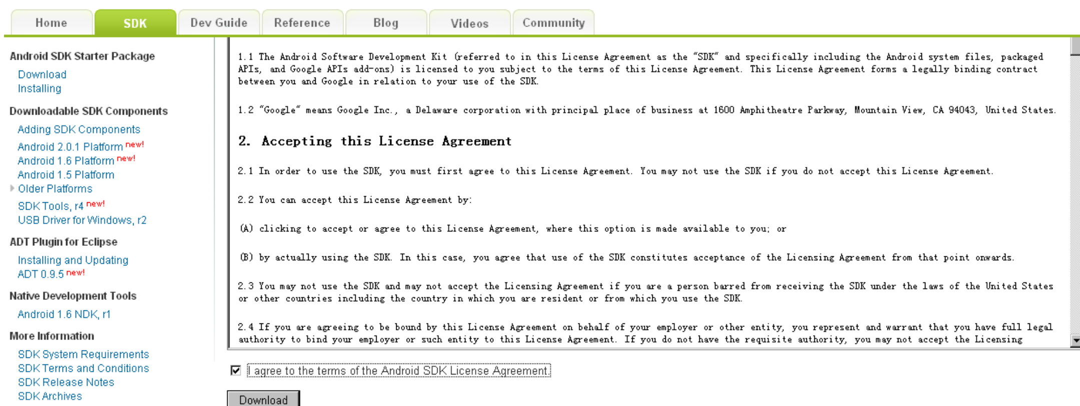


图 2-20　Android　SDK 下载页面

此时选中“I agree to the terms of the Android SDK License Agreement”选项，单击“Download”按钮开始下载。

第 3 步：下载完成后，解压压缩文件。假设下载后的文件解压存放在 F:\android\目录下，并将其 tools 目录的绝对路径添加到系统的路径中，具体操作方法如下。

1）用鼠标右键依次单击“我的电脑”→“属性”→“高级”，单击下面的“环境变量”，在下面的“系统变量”处选择“新建”选项，在变量名处输入 SDK_HOME，变量值中输入刚才的目录，比如“F:\android-sdk-windows”，如图 2-21 所示。

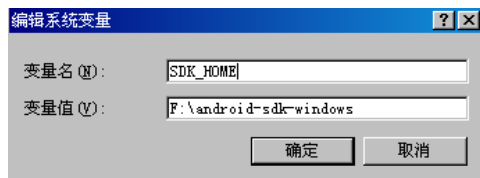


图 2-21　设置系统变量（1）

2）找到路径的变量，双击或单击编辑，在变量值最前面加上“%SDK_HOME%\tools;”。如图 2-22 所示。

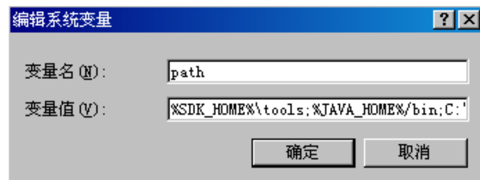


图 2-22　设置系统变量（2）

3）再次依次单击“开始”→“运行”，在运行框中输入“cmd”并按〈Enter〉键，在打开的 CMD 窗口中输入测试命令 android-h，如果显示如图 2-23 所示的提示信息，则说明安装成功。

```
C:\WINDOWS\system32\cmd.exe
C:\Documents and Settings\Administrator>android -h

Usage:
  android [global options] action [action options]

Global options:
  -v --verbose  Verbose mode: errors, warnings and informational messages are pr
inted.
  -h --help     This help.
  -s --silent   Silent mode: only errors are printed out.

Valid actions are composed of a verb and an optional direct object:
-   list              : Lists existing targets or virtual devices.
-   list avd          : Lists existing Android Virtual Devices.
-   list target       : Lists existing targets.
- create avd          : Creates a new Android Virtual Device.
-   move avd          : Moves or renames an Android Virtual Device.
- delete avd          : Deletes an Android Virtual Device.
- update avd          : Updates an Android Virtual Device to match the folders of
 a new SDK.
- create project      : Creates a new Android Project.
- update project      : Updates an Android Project (must have an AndroidManifest.
xml).
- create test-project: Creates a new Android Test Project.
- update test-project: Updates an Android Test Project (must have an AndroidMani
```

图 2-23　CMD 界面

4．安装 ADT

Android 为 Eclipse 定制了一个插件，即 Android Development Tools（ADT），这个插件为用户提供了一个强大的综合环境，用于开发 Android 应用程序。ADT 扩展了 Eclipse 的功能，可以让用户快速地建立 Android 项目，创建应用程序界面，在基于 Android 框架 API 的基础上添加组件，以及用 SDK 工具集调试应用程序，甚至导出签名（或未签名）的 APKs 以便发行应用程序。下面详细介绍安装配置 ADT 的基本方法。

要安装 Android Development Tools plug-in，需要首先打开 Eclipse IDE，然后进行如下操作。

第 1 步：打开 Eclipse 后，依次单击菜单栏的“Help”→“Install New Software...”，如图 2-24 所示。

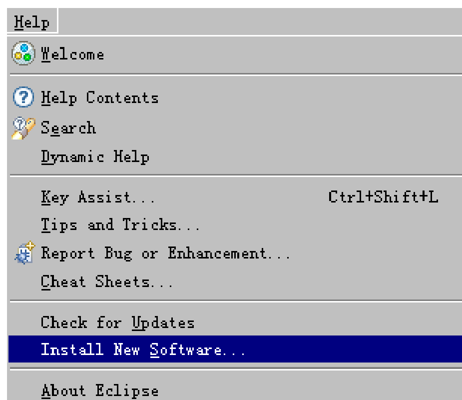


图 2-24　添加插件

第 2 步：在弹出的对话框中，单击“Add”按钮，如图 2-25 所示。



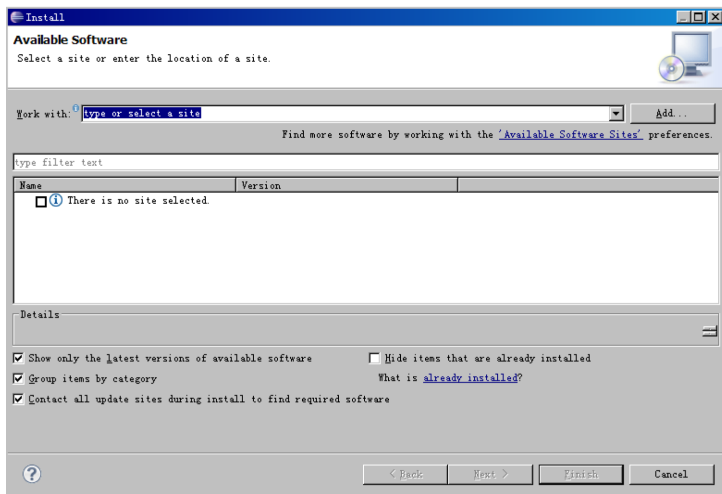


图 2-25　添加插件

第 3 步：在弹出的“Add Site”对话框中分别输入 Name 和 Location，Name 可以自己命名，如“guan”，但是 Location 中必须输入插件的网络地址“http://dl-ssl.google.com/Android/eclipse/”，单击“OK”按钮，如图 2-26 所示。

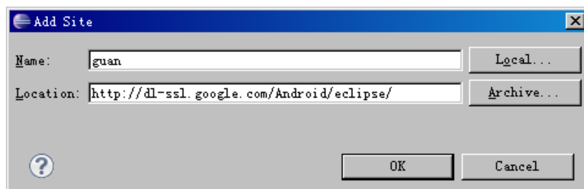


图 2-26　设置地址

第 4 步：单击“OK”按钮，完成设置，此时在“install”界面将会显示可用的插件。如图 2-27 所示。

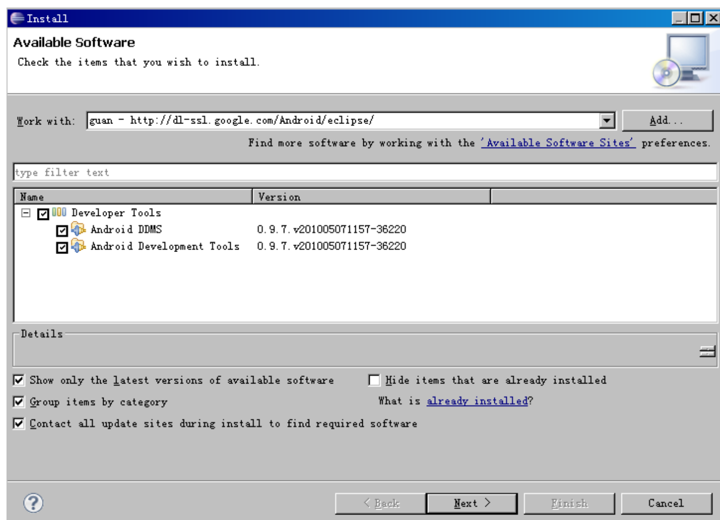


图 2-27　插件列表

第 5 步：把“Android DDMS”和“Android Development Tools”都选中，然后单击“Next”按钮，进入安装界面，如图 2-28 所示。

Install

Review Licenses

Licenses must be reviewed before the software can be installed. This includes licenses for software required to complete the install.

Items with licenses:

Name	Version
Android DDMS	0.9.5.v200911191123-20404
Android Development Tools	0.9.5.v200911191123-20404
Mylyn Bridge: Eclipse IDE	3.3.0.v20091015-0500-e3x
Mylyn Bridge: Java Development	3.3.0.v20091015-0500-e3x
Mylyn Bridge: Team Support	3.3.0.v20091015-0500-e3x
Mylyn Connector: Bugzilla	3.3.0.v20091015-0500-e3x
Mylyn Task List (Required)	3.3.0.v20091015-0500-e3x
Mylyn Task-Focused Interface (Rec...	3.3.0.v20091015-0500-e3x
Mylyn WikiText	1.2.0.v20091015-0500-e3x

License text:

Note: jcommon-1.0.12.jar is under the BSD license rather than the APL. You can find a copy of the BSD License at http://www.opensource.org/licenses/bsd-license.php

jfreechart-1.0.9.jar and jfreechart-1.0.9-swt.jar are under the LGPL rather than the APL. You can find a copy of the LGPL at http://www.gnu.org/licenses/old-licenses/lgpl-2.1.txt. You can get the source code for these two components at http://android.git.kernel.org/pub/jfreechart-1.0.9.zip

Apache License
Version 2.0, January 2004

http://www.apache.org/licenses/

TERMS AND CONDITIONS FOR USE, REPRODUCTION, AND DISTRIBUTION

I accept the terms of the license agreements
I do not accept the terms of the license agreements

< Back　Next >　Finish　Cancel

图 2-28　插件安装界面

第 6 步：此时选择“I accept the terms of the license agreement”选项，然后单击“Finish”按钮，则开始进行安装，如图 2-29 所示。

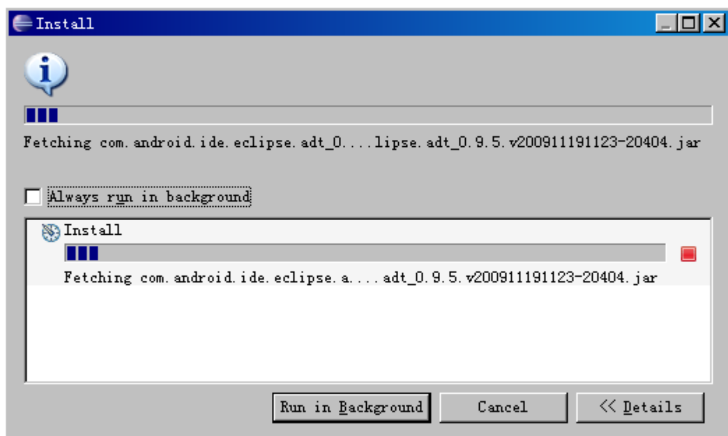


图 2-29　开始安装

注意：上述步骤可能速度有点慢，读者需要耐心，慢慢等待。完成后会提示重启 Eclipse 来加载插件，重启后就可以用了。

另外，不同版本的 Eclipse 安装插件的方法和步骤是不同的，但是都大同小异，相信读者参照上面的操作能够自行解决。



2.3.2　设定Android SDK Home

安装好插件后，还需要做如下配置才可以使用 Eclipse 创建 Android 项目，需要设置 Android SDK 的主目录。具体方法如下。

第 1 步：打开 Eclipse，在菜单中依次单击“Windows”→“Preferences”项，如图 2-30 所示。

第 2 步：在弹出的界面左侧可以看到“Android”项，选中 Android 后，在右侧设定 Android SDK 所在目录为 SDK Location，单击“OK”按钮完成安装，如图 2-31 所示。



图 2-30　Preferences 项

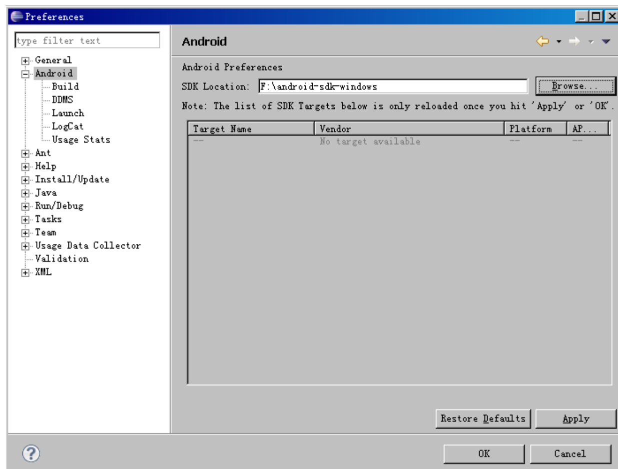


图 2-31　Preferences 项

2.3.3　开发环境验证

经过本章前面知识的讲解，Android 开发环境搭建完成。下面通过新建一个项目来验证当前的环境是否可以正常工作，具体流程如下。

第 1 步：打开 Eclipse，在菜单中依次选择“File”→“New”→“Project”项，在弹出的对话框上可以看到 Android，如图 2-32 所示。

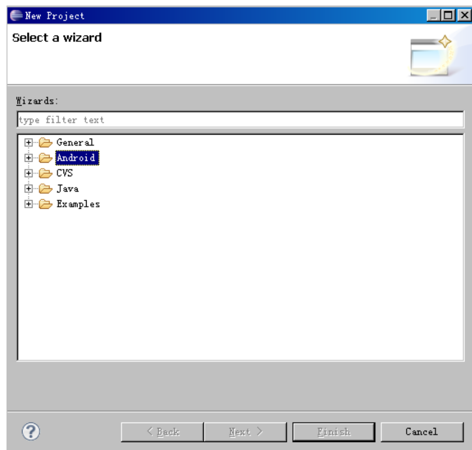


图 2-32　新建项目

第 2 步：在图 2-32 上选择“Android”，单击“Next”按钮，打开“New Android Project”对话框，在对应的文本框中输入必要的信息，如图 2-33 所示。

第 3 步：这里不再具体说明项目信息中各项的意义，后面章节会详细介绍。单击“Finish”按钮，会自动完成项目的创建工作，最后将可以看到如图 2-34 所示的项目结构。

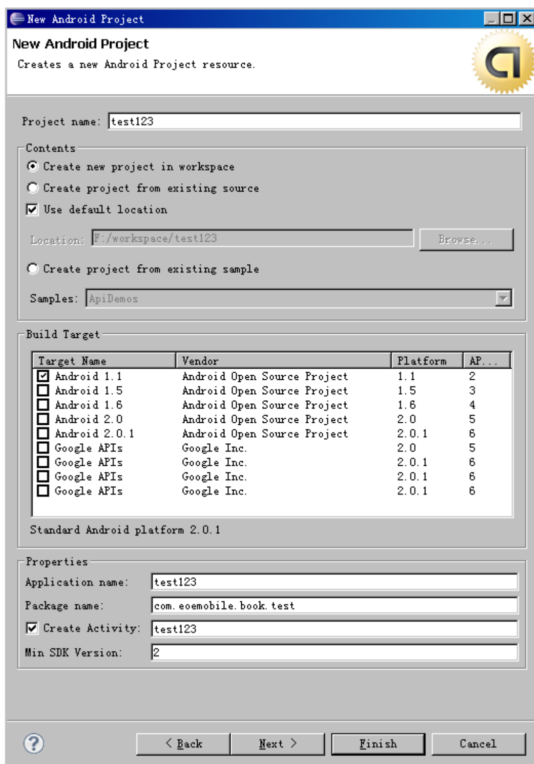


图 2-33 “New Android Project”对话框

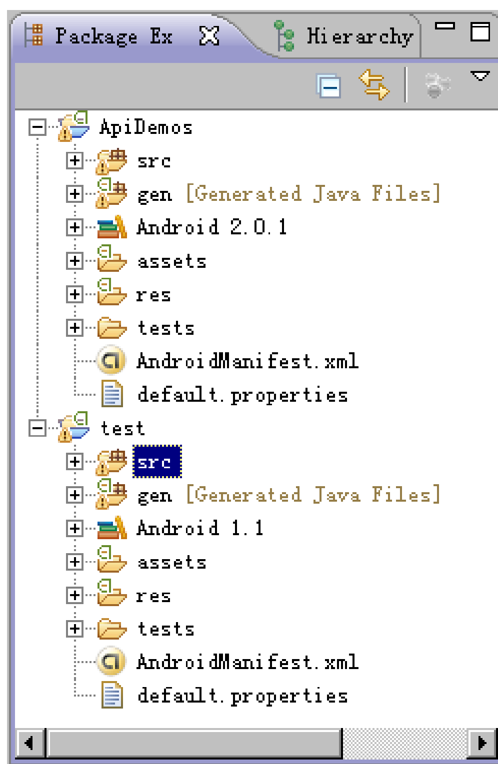


图 2-34 项目结构

通过上述操作，已经在 Windows 平台上搭建好了开发环境。

2.3.4 创建Android虚拟设备(AVD)

在 Android SDK1.5 版以后的 Android 开发中，必须创建至少一个 AVD，AVD 全称为 Android 虚拟设备（Android Virtual Device），每个 AVD 模拟了一套虚拟设备来运行 Android 平台，这个平台至少要有自己的内核，系统图像和数据分区，还可以有自己的 SD 卡和用户数据以及外观显示等。

因为 Android SDK1.5 以后支持多个平台和外观显示，作为开发者创建不同的 AVD 来模拟和测试不同的平台环境，创建 AVD 方法如下。

第 1 步：在 CMD 界面输入 android list targets 查看可用的平台，如图 2-35 所示。

如上列举了 7 个 targets，id 分别是 1、2、3、4、5、6、7。

第 2 步：创建 AVD。按照“android create avd --name <your_avd_name> --target <targetID>”格式创建 AVD，其中“your_avd_name”是需要创建的 AVD 的名字，在 CMD 窗口界面中如图 2-36 所示。



```
C:\WINDOWS\system32\cmd.exe
     Name: Android 2.0
     Type: Platform
     API level: 5
     Revision: 1
     Skins: HVGA (default), QVGA, WQVGA400, WQVGA432, WVGA800, WVGA854
id: 5 or "android-6"
     Name: Android 2.0.1
     Type: Platform
     API level: 6
     Revision: 1
     Skins: HVGA (default), QVGA, WQVGA400, WQVGA432, WVGA800, WVGA854
id: 6 or "android-7"
     Name: Android 2.1-update1
     Type: Platform
     API level: 7
     Revision: 2
     Skins: HVGA (default), QVGA, WQVGA400, WQVGA432, WVGA800, WVGA854
id: 7 or "android-8"
     Name: Android 2.2
     Type: Platform
     API level: 8
     Revision: 1
     Skins: HVGA (default), QVGA, WQVGA400, WQVGA432, WVGA800, WVGA854

C:\Documents and Settings\Administrator>
```

图 2-35　CMD 界面

```
C:\WINDOWS\system32\cmd.exe
Microsoft Windows XP [版本 5.1.2600]
(C) 版权所有 1985-2001 Microsoft Corp.

C:\Documents and Settings\Administrator>android create avd --ff --target 9_
```

图 2-36　CMD 界面

第 3 步：这样就创建了一个自定义的 AVD(Android Virtual Device)。然后，只要在 Eclipse 的 Run Configurations 里面指定一个 AVD，即在 Target 下选中自己定义的这个 AVD，即 sdk_2_3_version 就可以运行了，如图 2-37 所示。

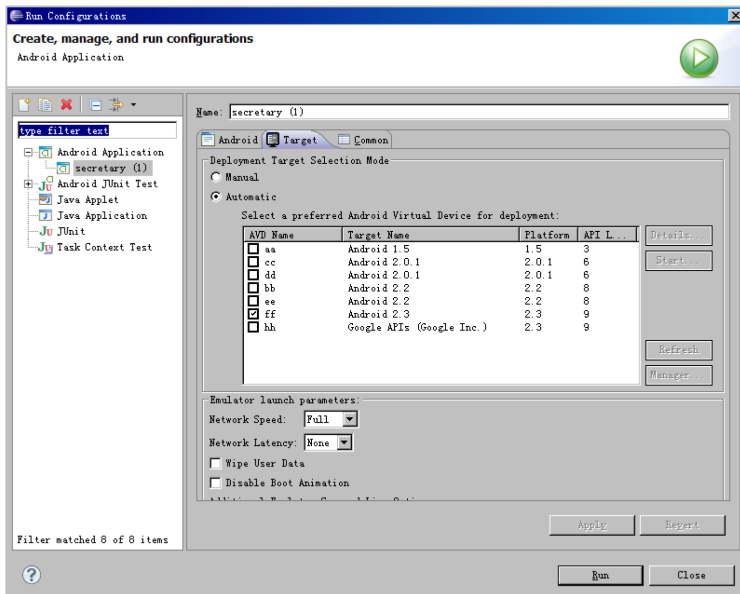


图 2-37　选择 AVD

第 4 步：如上我们选择“sdk_2_3_version”，单击“Apply”按钮后，然后单击“Start”按钮弹出“Launch”对话框，如图 2-38 所示。

第 5 步：此时单击“Launch”按钮后将会运行模拟器，如图 2-39 所示。

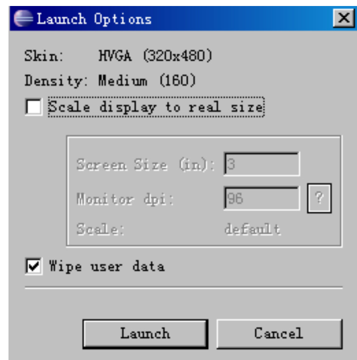


图 2-38 “Launch”对话框

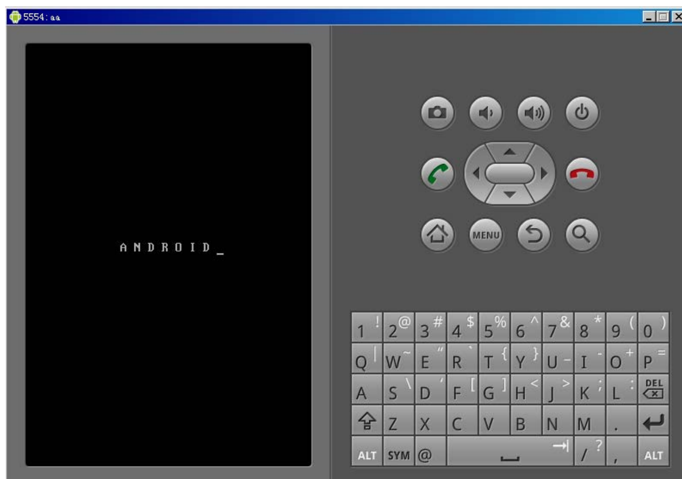


图 2-39 模拟运行成功

关于应用结构分析和讲解，以及代码的调试部分内容会在本书后面的内容中进行详细介绍。至此，在 Windows 平台上的开发环境搭建完成，安装了运行环境 JDK，开发工具 Eclipse，Android SDK 和 ADT 并进行 SDK Home 的配置，最后创建了一个 Android 虚拟设备（AVD）。

2.4 常见的一些问题

安装和搭建一个开发环境，常常会遇到一些意想不到的问题，这些问题可能是我们粗心造成的，也可能是使用的系统环境的差异造成的。在本节内容中，将简单介绍搭建 Android 开发环境中常见问题的解决方法。

1. Android不能在线更新

在安装 Android 后，需要更新为最新的资源和配置。但是在启动 Android 后，经常会不能更新，弹出如图 2-40 所示的错误提示。

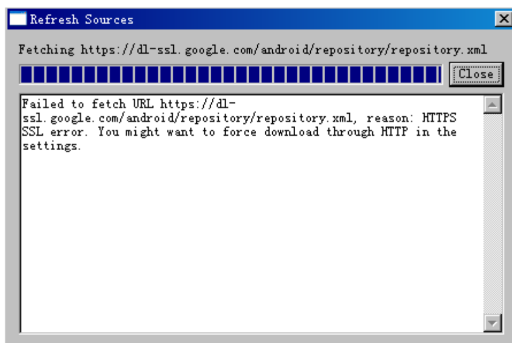


图 2-40 不能更新



Android 默认的在线更新地址是 https://dl-ssl.google.com/android/eclipse/，但是经常会出现错误。如果此地址不能更新，可以自行设置更新地址，修改为 http://dl-ssl.google.com/ android/ repository/repository.xml，具体操作方法如下。

1）单击 Android 左侧的“Available Packages”选项，然后单击下面的“Add Site…”按钮。如图 2-41 所示。

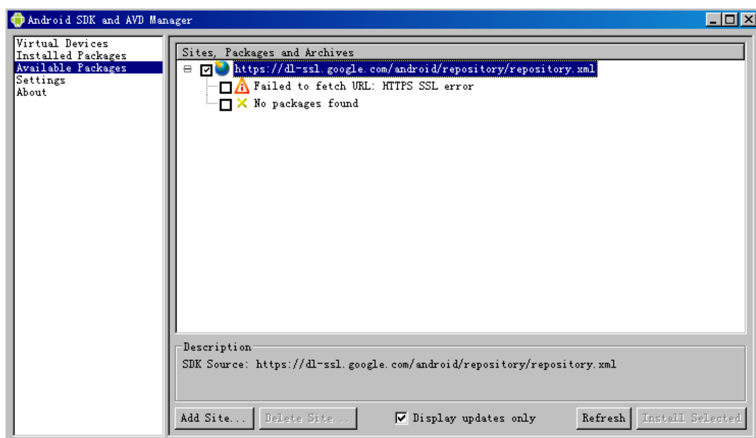


图 2-41　“Available Packages”界面

2）在弹出的“Add Site URL”对话框中输入修改后的地址 http://dl-ssl.google.com/ android/ repository/repository.xml，如图 2-42 所示。

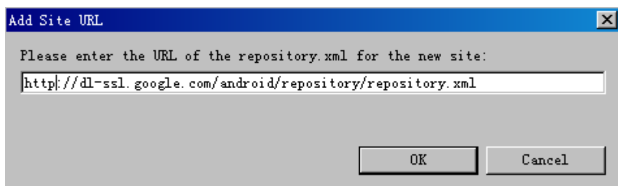


图 2-42　“Add Site URL”界面

3）单击“OK”按钮，完成设置。经过上述操作后，就能够使用更新功能了，如图 2-43 所示。

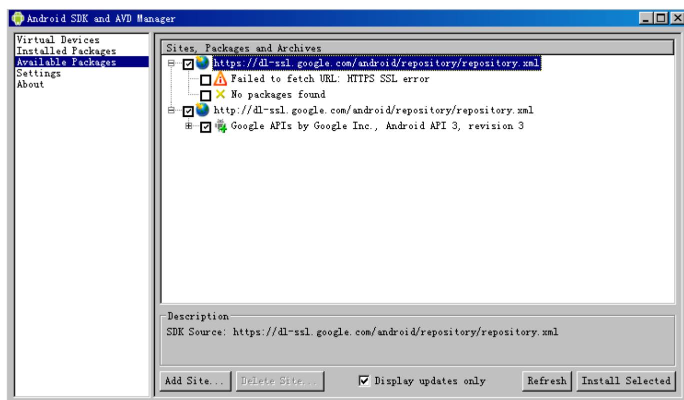


图 2-43　“Available Packages”界面

2. Eclipse中新建Android工程时出错

在 Eclipse 中新建 Android 工程时，一直显示 Project name must be specified，如图 2-44 所示。

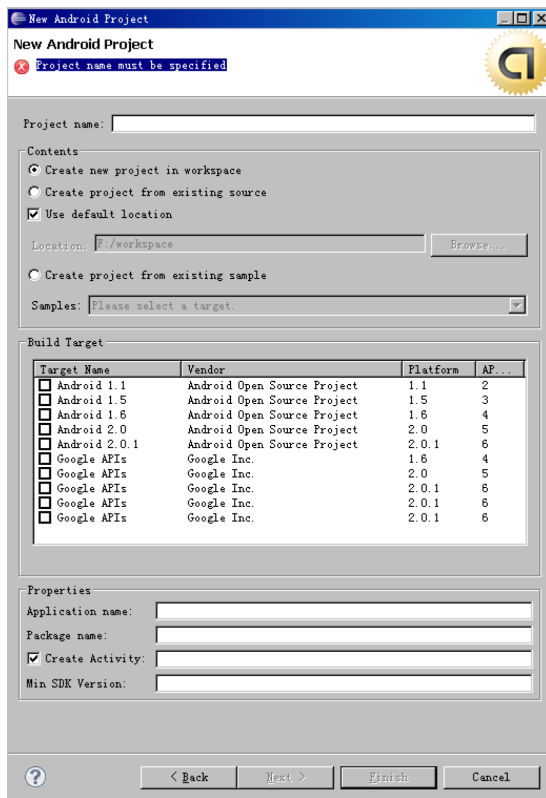


图 2-44 “New Android Project”界面

造成上述问题的原因是 Android 没有更新完成，需要进行完全更新。具体方法如下。

1）打开 Android，选择左侧的“Installed Packages”，如图 2-45 所示。

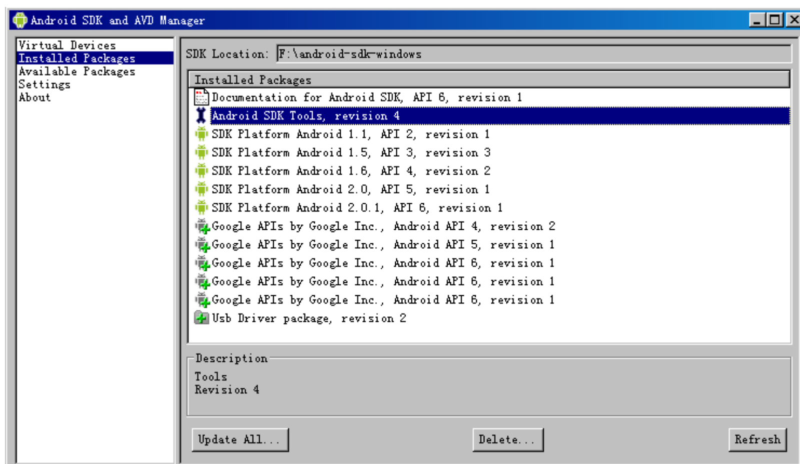


图 2-45 “Available Packages”界面



2）右侧列表中选择要升级的选项，例如，选择“Android SDK Tools ,revision4　”，在弹出窗口中先选择“Accept”，然后单击“Install Accepted”按钮开始安装更新，如图 2-46 所示。

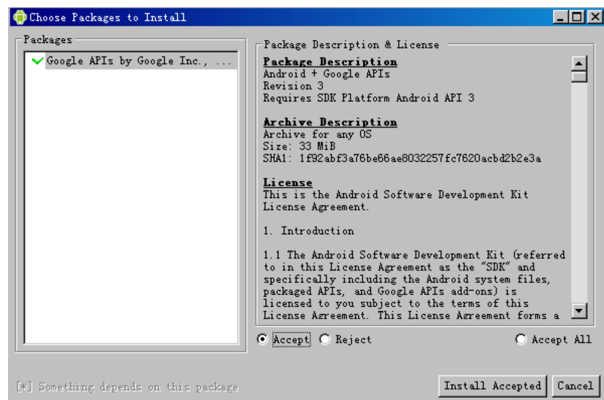


图 2-46　“Choose Packages Install”界面

3. Eclipse 中没有 Target 选项

通常来说，当 Android 开发环境搭建完毕后，会在“Preference”中显示存在的 SDK Targets，如图 2-47 所示。

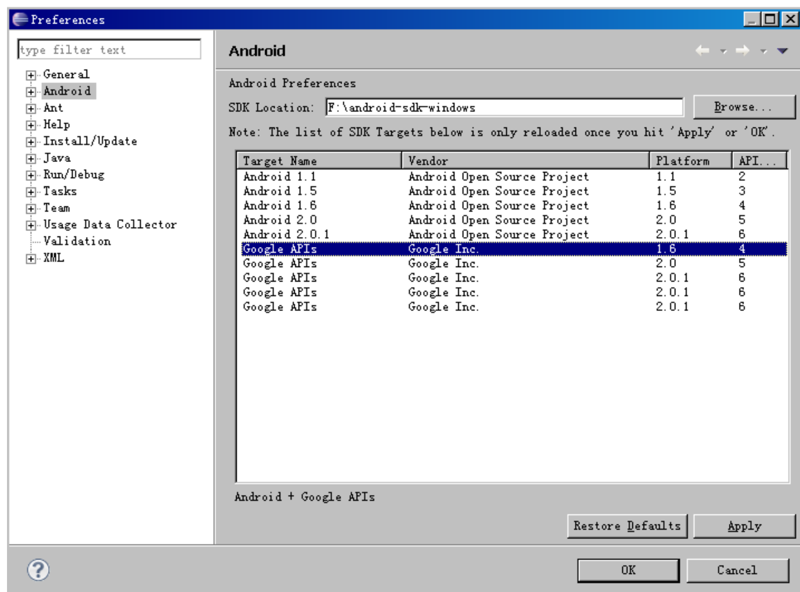


图 2-47　SDK Targets 列表

但是往往因为各种原因，会不显示 SDK Targets 列表，并且在图 2-33 界面中也不显示，并输出“Failed to find an AVD compatible with target”提示的错误问题。上述问题是 AVD 没有创建成功，如果出现上述问题，就需要我们手工安装，当然前提是 Android 更新完毕。具体解决方法如下。

1）在运行中键入“cmd”，打开 CMD 窗口，如图 2-48 所示。

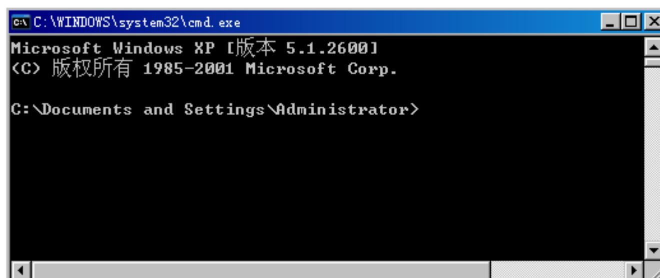


图 2-48　SDK Targets 列表

2）使用 Android 命令创建一个 Targets，创建 AVD。按照“android create avd --name <your_avd_name> --target <targetID>”格式创建 AVD，其中“your_avd_name”是需要创建的 AVD 的名字，如图 2-49 所示。

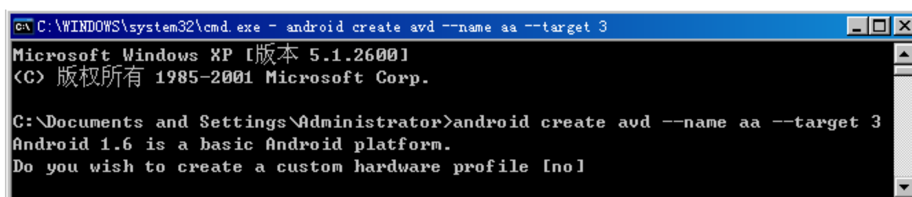


图 2-49　CMD 界面

在图 2-49 的窗口中创建了一个名为 aa，targetID 为 3 的 AVD，然后在 CMD 界面中输入“n”，即完成操作，如图 2-50 所示。

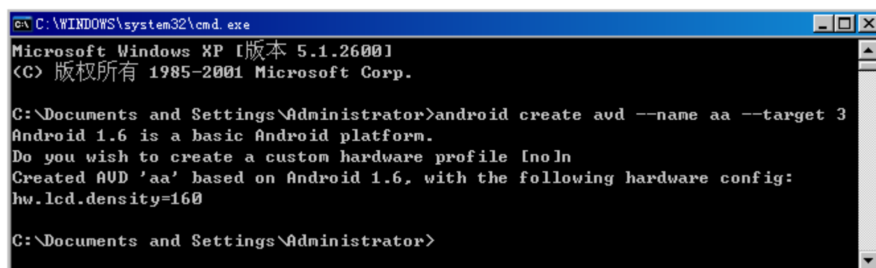


图 2-50　CMD 界面

4. 在线更新时出错

在线更新时，提示“Failed to rename directory d:\android-sdk-windows\tools to d:\android-sdk-windows\tools\temp\ToolPackage.old01”之类的错误。

通常是因为杀毒软件或防火墙对这个文件进行了控制，建议读者在更新时关闭杀毒软件和防火墙。如果还不行则进行如下设置。

1）在 android-sdk-windows 下复制 tools 文件夹，得“复件 tools”文件夹。

2）运行复件 tools 文件夹里的 android.bat，在出现的 Android SDK and AVD Manager 上执行之前的升级步骤即可成功。记得在此步之前关闭已打开的 Android SDK.. Manager（若有）。

3）删除“复件 tools”（非必要）。

第 3 章　Android SDK简要介绍

前面的内容已经介绍了 Android SDK 的安装知识，为读者进行 Android 开发打好了基础。在本章的内容中，将对 Android SDK 集成开发环境进行介绍，以便于读者理解本书后面实例的开发过程。

3.1　Android SDK基础

与 iPhone IOS 相似，Android 系统采用 WebKit 浏览器引擎，具备触摸屏、高级图形显示和上网功能，用户能够在手机上查看电子邮件、搜索网址和观看视频节目等。Android 系统比其他手机系统更强调搜索功能，界面更强大，可以说是一种融入全部 Web 应用的平台。

Android SDK 以 Java 语言为基础，用户可以使用 Java 来开发 Android 平台上的应用软件。通过 SDK 提供的一些工具将其打包成 Android 平台使用的 apk 文件，然后再使用 SDK 中的模拟器来模拟和测试该软件在 Android 上的运行效果。

3.2　初步探寻Android SDK体系

3.2.1　Android SDK目录结构

安装 Android SDK 后，打开安装目录，其具体目录结构如图 3-1 所示。

- add-ons：里面包含了官方提供的 API 包，最主要的是 Map 的 API 文件。
- docs：里面包含了文档，即帮助文档和说明文档。
- platforms：针对每个版本的 SDK 版本提供了和其对应的 API 包以及一些示例文件，其中包含了各个版本的 Android，如图 3-2 所示。
- temp：里面包含了一些常用的文件模板。
- tools：包含了一些通用的工具文件。
- usb_driver：包含了 AMD64 和 X86 下的驱动文件。
- SDK Setup.exe：Android 的启动文件。

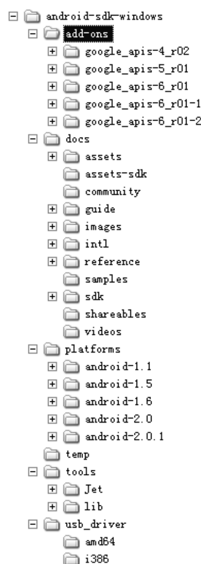


图 3-1　Android SDK 安装后的目录结构



图 3-2　platforms 目录项

3.2.2　android.jar及内部结构

在 platforms 目录下的每个 android 版本中，都有一个 android.jar 文件。例如“platforms\android-8”下的如图 3-3 所示。

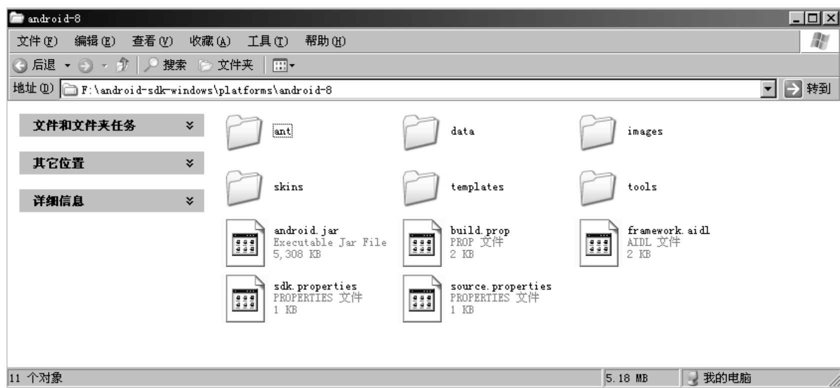


图 3-3　android.jar 文件所在目录

android.jar 文件是一个标准的压缩包，里面包含了编译后的压缩文件，使用 Windows 系统上的解压缩工具 WinRAR 可以打开此压缩文件，此时可以看到其内部结构如图 3-4、图 3-5 所示。



图 3-4　android.jar 文件结构





图 3-5 “android.jar.android\agg”文件结构

从图 3-4、图 3-5 所示的结构可以看出，它对 API 包进行了详细的划分，分为了 app、content、database 等。用户只要理解了其模块划分结构，就可以通过 SDK 文档了解其具体信息。

3.2.3 SDK文档及阅读技巧

如果要深入理解各个文件包内包含的 API 和 API 的具体用法，就必须学会阅读、查找 SDK 文档。

可以使用浏览器打开“docs”目录下的 index.html，如图 3-6 所示。

在图 3-6 所示的主页中，介绍了 Android 基本概念、当前版本，在右侧和顶端导航中列出了一些常用的链接。这个 SDK 文件对初学者来说十分重要，可以帮助读者解决很多常见的问题，并介绍了 Android 的基本知识，可以说是一个很好的学习文档、帮助文档。

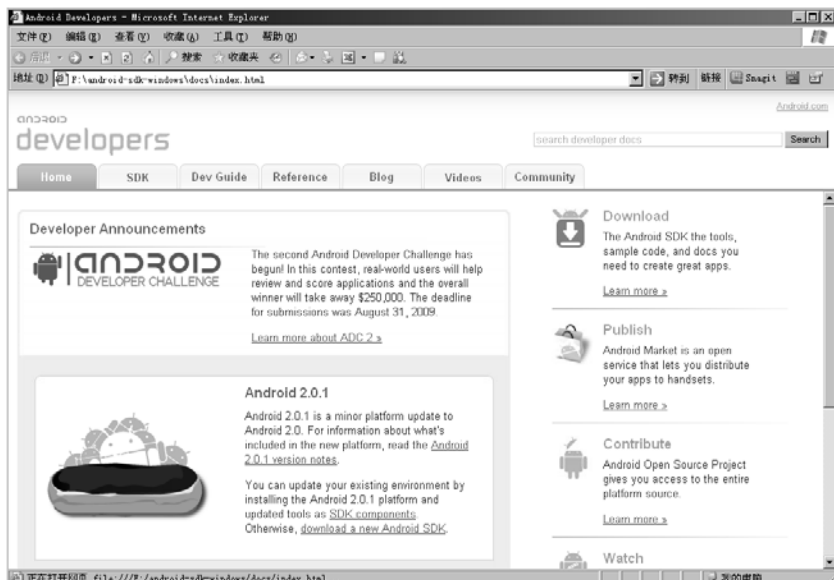


图 3-6 SDK 文档主页

单击导航中的“Dev Guide”按钮，打开如图 3-7 所示界面。



图 3-7　SDK 文档索引

图 3-7 所示页面中，左侧是目录索引链接，单击某个链接后，可以在右侧界面中显示对应的说明信息。下面对各个索引目录链接进行简单的介绍。

如果要想迅速地理解一个问题或知识点，可以在搜索对话框中对 SDK 进行检索，搜索到自己需要的内容。当然，很多热心的程序员和学者对 SDK 进行了翻译，网络上有很多 SDK 中文版，感兴趣的读者可以从网络中获取。

3.2.4　SDK工具集

在 SDK 中，集成了很多有用的开发工具，这些工具能够帮助读者在 Android 平台上开发出应用程序。在 Android SDK 中，最有用的是 Android 模拟器和 Eclipse 的 Android 开发插件，但是 SDK 中也包含了各种在模拟器上用于调试、打包和安装的工具。下面将简要介绍这些工具的基本使用知识。

❑ Android 模拟器

Android 模拟器是运行在计算机上的虚拟移动设备，有关模拟器的基本知识已经在本书的第一章中进行了详细介绍，在此不再讲解。

❑ 集成开发插件 ADT

Android 为 Eclipse 定制了一个插件，即 Android Development Tools（ADT），这个插件为用户提供一个强大的综合环境用于开发 Android 应用程序。ADT 扩展了 Eclipse 的功能，可以让用户快速地建立 Android 项目，创建应用程序界面，在基于 Android 框架 API 的基础上添加组件，以及用 SDK 工具集调试应用程序，甚至导出签名（或未签名）的 APKs 以便发行应用程序。

❑ 调试监视服务 ddms.bat



调试监视服务 ddms.bat 集成在 Dalvik（Android 平台的虚拟机）中，用于管理运行在模拟器或设备上的进程，并协助调试工作。它可以去除一些进程，选择一个特定的程序来调试，生成跟踪数据，查看堆和线程数据，对模拟器或设备进行屏幕快照等操作。

❑ Android 调试桥 adb.exe

Android 调试桥(adb)是多种用途的工具，该工具可以帮助用户管理设备或模拟器的状态。

可以通过下列几种方法加入 adb。

1）在设备上运行 shell 命令。

2）通过端口转发来管理模拟器或设备。

3）从模拟器或设备上复制来或复制走文件。

❑ Android 资源打包工具 aapt.exe

可以通过 aapt.exe 工具来创建 apk 文件，在 apk 文件中包含了 Android 应用程序的二进制文件和资源文件。

❑ Android 接口描述语言 aidl.exe

用于生成进程间接口代码。

❑ SQLite3 数据库 sqlite3.exe

Android 可以创建和使用 SQLite 数据文件，开发人员和用户可以方便地访问这些 SQLite 数据文件。

❑ 跟踪显示工具

可以生成跟踪日志数据的图形分析视图，这些跟踪日志数据由 Android 应用程序产生。

❑ 创建 SD 卡工具

用于创建磁盘镜像，此镜像可以在模拟器上模拟外部存储卡，如常见的 SD 卡。

❑ DX 工具（dx.bat）

将 class 字节码重写为 Android 字节码（被存储在 dex 文件中）。

❑ 生成 Ant 构建文件（activitycreator.bat）

activitycreator.bat 是一个脚本，用于生成 Ant 构建文件。Ant 构建文件用于编译 Android 应用程序，如果在安装 ADT 插件的 Eclipse 环境下开发，就不需要这个脚本了。

❑ Android 虚拟设备

在 Android SDK 1.5 版以后的 Android 开发中，必须创建至少一个 AVD，AVD 全称为 Android 虚拟设备（Android Virtual Device），每个 AVD 模拟了一套虚拟设备来运行 Android 平台，这个平台至少要有自己的内核，系统图像和数据分区，还可以有自己的 SD 卡和用户数据以及外观显示等。

在本书第 2 章中已经介绍了 Android 虚拟设备的安装方法，在此将不再具体介绍。

3.3 解析Android SDK实例

在 Android 的目录中有一个名为“samples”的文件夹，下面存放了 SDK 中的几个演示实例。这些实例从不同的方面展示了 SDK 的特性。例如“android-3”中的实例文件夹如图 3-8 所示。

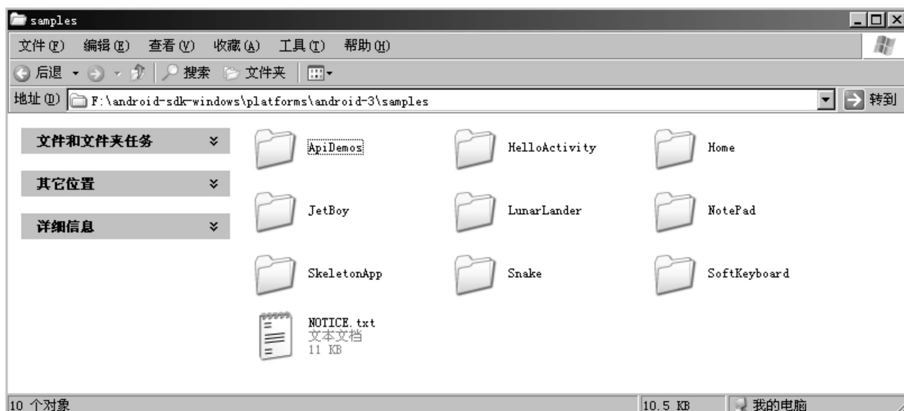


图 3-8　演示实例结构

1．HelloActivity

这和其他编程语言中的 Hello World 程序类似，是一个 Android 平台上的最简单程序，运行后将在手机上显示出“Hello World”的提示。打开 Eclipse，将“HelloActivity”导入，然后查看执行后的效果，具体如图 3-9 所示。

注意：在查看安装目录中的“samples”实例时，不能使用“Import”将实例导入到 Eclipse 中。要查看实例的运行效果，需要按照下面的步骤操作。

1）Eclipse 中依次单击“file”→“new”→“android project”，弹出“new android project”对话框。在里面选择“create project from existing source”选项，然后单击“Browse”按钮，并选择对应的实例文件夹即可，如图 3-10 所示。



图 3-9　执行效果（1）

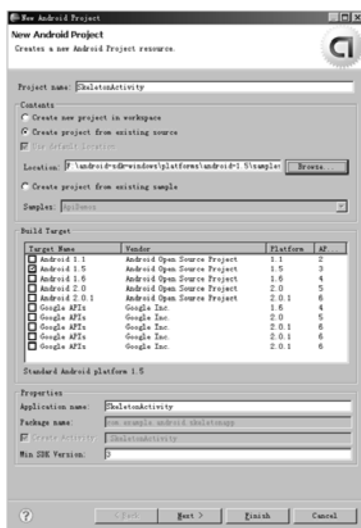


图 3-10　“New Android Project”对话框

2）单击“Finish”按钮完成操作，这样即可将实例程序成功导入到 Eclipse 中。

2．视图组件SkeletonApp

本实例展示了如何在 Android 中应用提供的视图组件，如常见的 EditText、Button、



ImageView 和菜单等，还演示了如何操作这些组件。执行后的效果如图 3-11 所示。

3．API应用实例ApiDemos

ApiDemos 演示了很多 API 的使用方法，包括 app、content、graphic、media 等，如图 3-12 所示。



图 3-11　执行效果（2）



图 3-12　执行效果（3）

在图 3-12 中可以选择查看具体的分类，进一步了解 API 的强大功能。

4．LunarLander

这是一个登月游戏实例，演示了一个类似于登陆月球的小游戏，可以通过方向键和点火时机控制画面上的飞船，如图 3-13 所示。

5．NotePad

NotePad 是一个记事本程序，此程序可以实现新建、编辑和删除等文档操作。本实例应用了 SQLite 的数据存储和编辑，并使用了 ContentProvider 等方面的信息。执行后效果如图 3-14 所示。



图 3-13　执行效果（4）

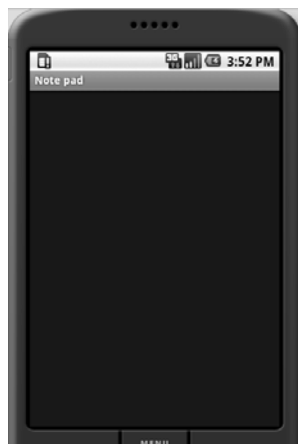


图 3-14　执行效果（5）

6．Snake

Snake 是贪吃蛇演示实例，这是一款经典的游戏，使用手机方向键可以对游戏进行控制。

执行后效果如图 3-15 所示。

7．Home

Home 是一款主题类软件实现的实例，实现了一套新的主题界面。此实例演示了如何开发主题类应用，读者通过这个实例可以轻松掌握主题类开发的步骤和一些注意事项。执行后效果如图 3-16 所示。

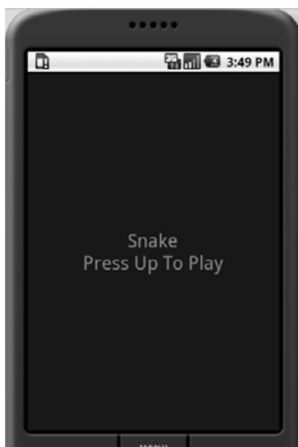


图 3-15　执行效果（6）



图 3-16　执行效果（7）

8．SoftKeyboard

SoftKeyboard 是一个软键盘实例，此实例演示了如何将软键盘绑定到输入框输入事件上。当焦点移到输入框上时，将自动显示软键盘。执行后效果如图 3-17 所示。

9．JetBoy

JetBoy 是一款具备声音支持的游戏实例，它模拟演示了如何在游戏中集成 SONiVOX 的 audioINSIDE 技术，此技术是 SONiVOX 捐赠给手机联盟的。此实例可以完美地播放背景音乐和场景，实现子弹击碎飞来障碍物等一系列的效果。执行后效果如图 3-18 所示。



图 3-17　执行效果（8）



图 3-18　执行效果（9）

至此，Android 安装目录中自带的实例文件介绍完毕。读者课后要仔细品味每个实例的具体效果，并尝试阅读每个实例的具体实现代码，为学习本书后面的知识打下基础。

第 4 章　看第一个应用程序

在本书前面的章节中，已经讲解了 Android 开发环境的搭建过程和 Android SDK 框架的基本知识。在接下来的内容中，要讲解 Android 应用程序的设计过程。本章将通过一个简单的 Android 实例，来说明 Android SDK 代码编写的具体流程。

4.1　Hello World应用程序分析

本章实例的功能是在屏幕中输出“Hello World”这段文字。在下面的内容中，将讲解本实例的具体实现过程。

4.1.1　新建一个Android工程

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Project”命令，新建一个工程文件，如图 4-1 所示。

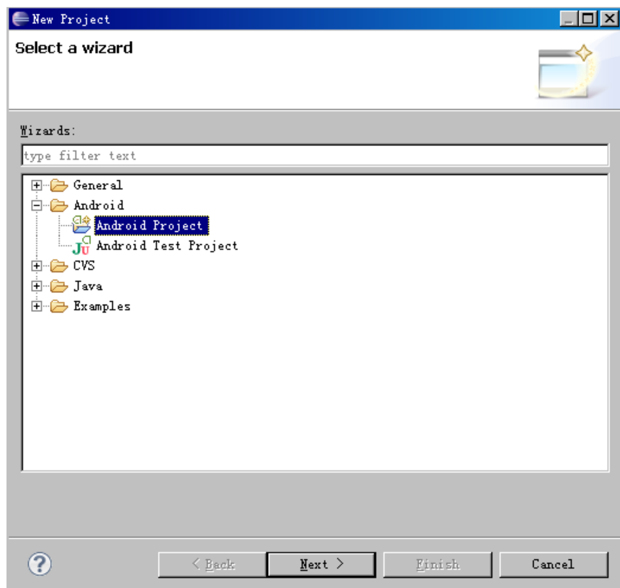


图 4-1　新建工程文件

第 2 步：选择“Android Project”选项，单击“Next”按钮。

4.1.2　设置工程的信息

在弹出的“New Android Project”对话框中，设置工程信息，如图 4-2 所示。

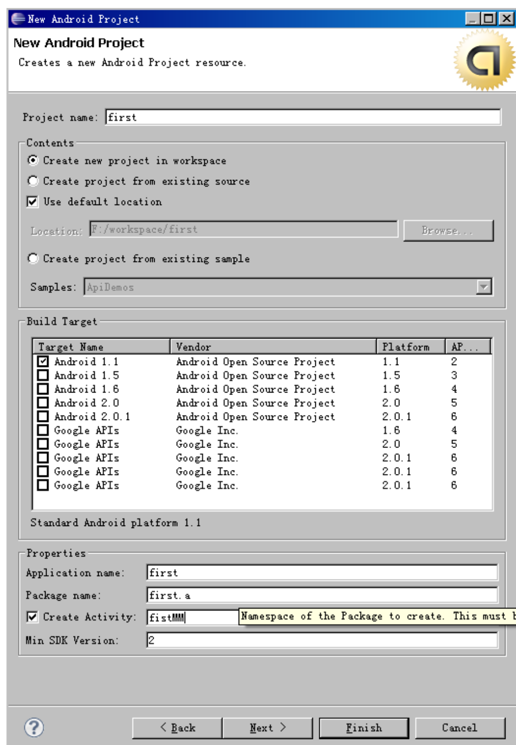


图 4-2　设置工程

在图 4-2 所示的界面中依次设置工程名字、包名字、Activity 名字、应用名字。

4.1.3　编写代码和代码分析

经过上述设置后，创建了一个名为“first”的工程文件。此时打开文件“fistMM.java”，会显示如下代码。

```
package first.a;
import android.app.Activity;
import android.os.Bundle;
public class fistMM extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

如果此时运行程序，将不会显示任何东西。用户可以对上述代码进行稍微的修改，让程序输出“Hello World”。具体代码如下。

```
package first.a;
import android.app.Activity;
```



```
import android.os.Bundle;
import android.widget.TextView;

public class fistMM extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        TextView tv = new TextView(this);
        tv.setText("HelloWorld");
        setContentView(tv);
    }
}
```

经过上述代码改写后，即可在屏幕中输出“Hello World”。

在 Android 项目中，几乎所有的 UI 都是由 View（视图）或 View（视图）子类实现的。View 代表了一块区域，此区域不但可以处理事件，并且可以渲染这块区域。在上述代码中使用了 TextView 组建，此组件是继承于 View 来实现文字显示功能的。为此可以认为，TextView 代表了一块区域，区域里面有一些文字信息，并且用户可以对区域里的文字进行修饰处理。

4.1.4　运行项目

将上述代码保存，即可运行这段程序了。具体过程如下。

第 1 步：右键单击项目名，在弹出命令中依次选择“Run As”→“Android Application”选项，如图 4-3 所示。

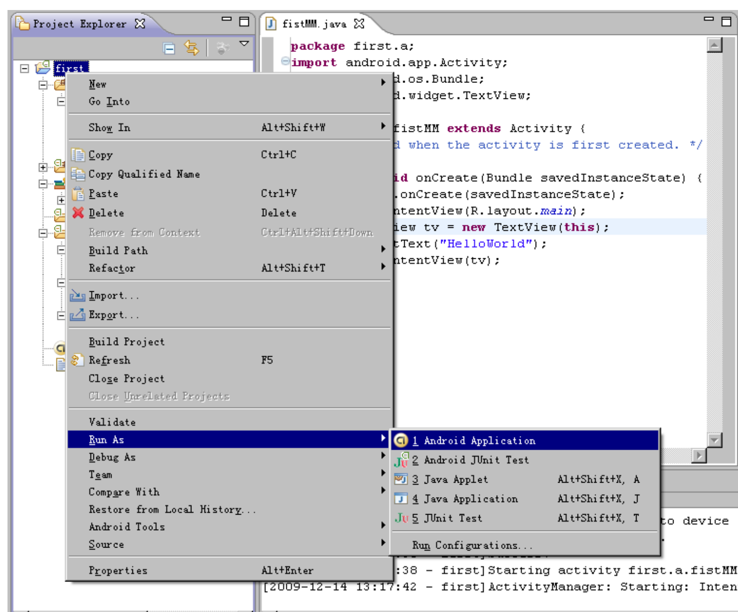


图 4-3　开始调试

第 2 步：此时工程开始运行，运行完成后在屏幕中输出“Hello World”这段文字，如图 4-4 所示。



图 4-4　运行结果

经过上述操作，即可成功显示图 4-5 所示的运行效果。

注意：如果加载的时间太长，运行后可能会一直显示图 4-4 所示的界面。解决上述问题的方法是：单击图 4-4 中的“MENUE”键，然后在弹出窗口中单击“Wait”按钮，如图 4-6 所示。

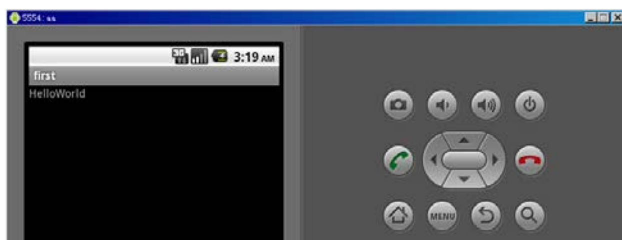


图 4-5　执行效果

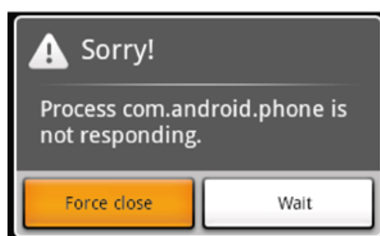


图 4-6　单击“Wait”按钮

4.2　调试项目

用 Android 开发一个项目后，可以对其进行调试处理。利用 Eclipse、Android 和另外的插件，可以在 Eclipse 中对 Android 程序进行断点调试。在本节的内容中，将简单介绍 Android 项目调试的基本知识。

4.2.1　设置断点

此处的设置断点和 Java 中的一样，可以通过双击代码左边的区域进行断点设置，如图 4-7 所示。为了调试方便，可以设置显示代码的行数。方法是在代码左侧的空白部分单击右键，在弹出命令中选择“show line numbers”，如图 4-8 所示。



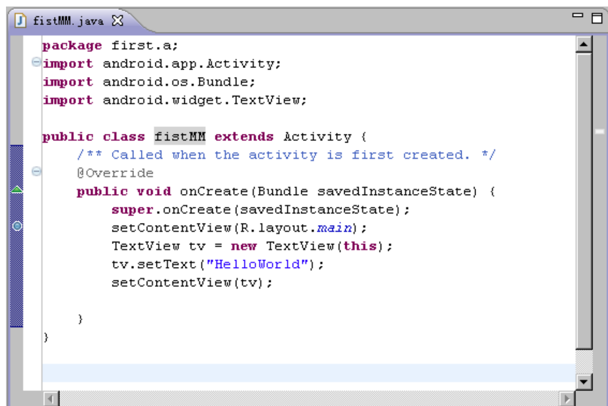


图 4-7　设置断点

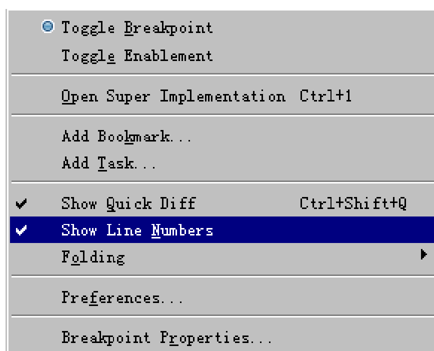


图 4-8　显示行数

4.2.2　Debug项目

Debug Android 项目操作和普通的 Debug Java 项目类似，只是在选择调试项目时选择“Android Application”即可。即右键单击项目名，在弹出命令中依次选择“Debug As”→“Android Application”选项，如图 4-9 所示。

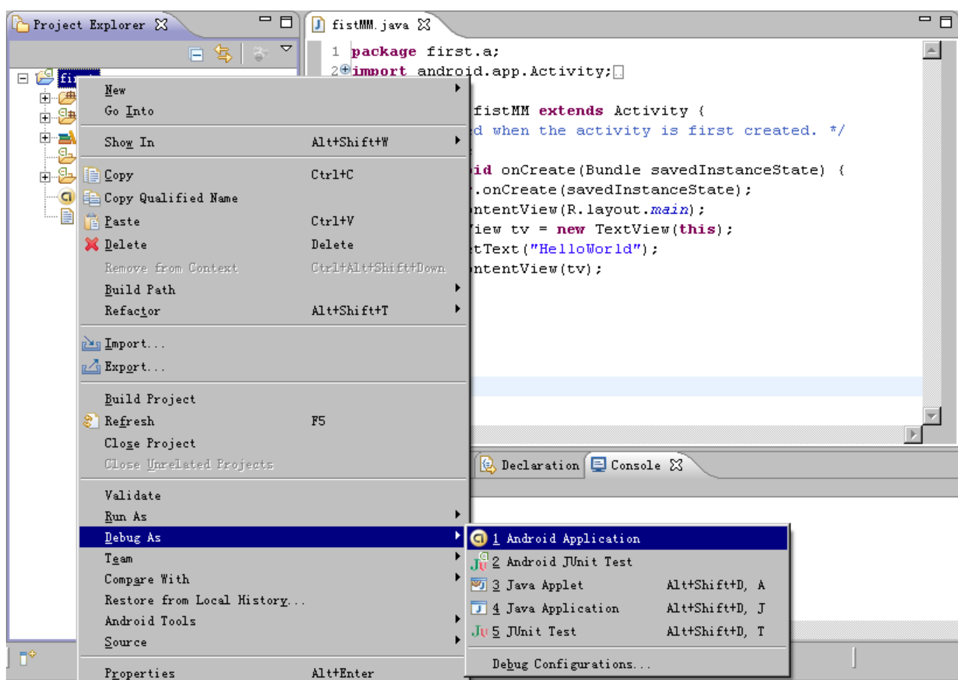


图 4-9　Debug 项目

4.2.3　断点调试

可以进行单步调试，具体调试方法和普通的 Java 程序类似，具体调试界面如图 4-10 所示。

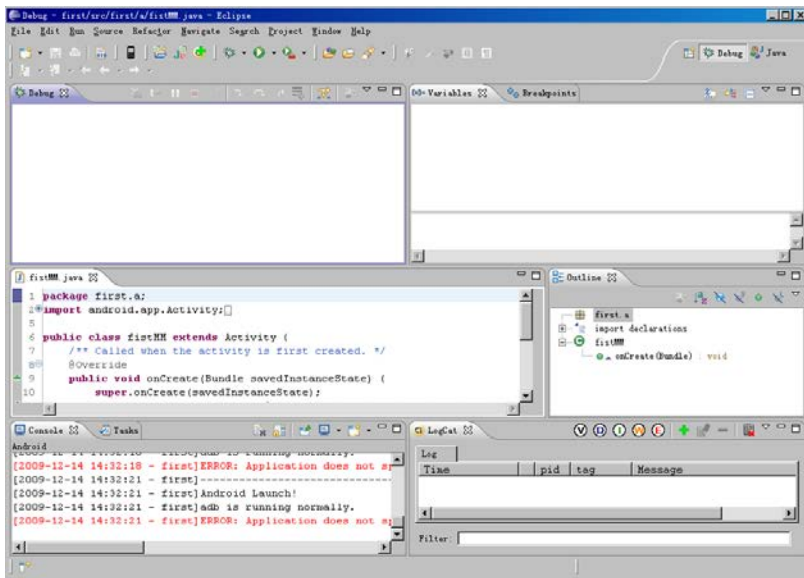


图 4-10　调试界面

4.3　Dialog（对话框）简介

对话框（Dialog）是桌面和 Web 应用程序中通用的 UI 称谓。它们用于帮助用户回答问题、选择选项、确认动作、阅读警告与错误消息。

4.3.1　Android对话框

Android 对话框是一个漂浮的窗口，启动它的 Activity（活动）会出现部分模糊。正如在图 4-11 中看到的，对话框不是全屏且可以部分透明。它们一般使用模糊或暗淡过滤器来模糊它们背后的 Activity。



图 4-11　手机中的对话框



在 Android 中，有 3 种方式来实现对话框。

1）使用 Dialog 类的后代。和一般意义的 AlertDialog 类一样，Android 包含多个扩展了 Dialog 类的特定类。每个被设计用来提供特定的对话框功能。基于 Dialog 类的屏幕全部在调用它们的 Activity 中创建，所以，用户不需要在 manifest 中注册，而且它们的生命周期完全由调用的 Activity 控制。

2）Dialog 主题的 Activity。可以应用 Dialog 主题到正常的 Activity 上，让它拥有和对话框一样的外观。

3）Toast（提醒）。Toast 是种特殊的、非模态的、短暂的消息对话框，通常在 Broadcast Receiver 和后台 Service 中使用，用来提示用户事件。

4.3.2　Dialog类详解

Dialog 类实现为一个简单的漂浮窗口，完全在 Activity 中创建。用户使用基本的 Dialog 类，可以创建一个新的实例并设定标题和布局，如下所示。

```
Dialog d = new Dialog(MyActivity.this);
// Have the new window tint and blur the window it
// obscures.
Window window = d.getWindow();
window.setFlags(WindowManager.LayoutParams.FLAG_BLUR_BEHIND,
WindowManager.LayoutParams.FLAG_BLUR_BEHIND);
d.setTitle(“Dialog Title”);
d.setContentView(R.layout.dialog_view);
TextView text = (TextView)d.findViewById(R.id.dialogTextView);
text.setText(“This is the text in my dialog”);
```

一旦按照需求完成配置后，只要使用下面的方法即可显示它。

```
d.show();
```

1．AlertDialog类

AlertDialog（提醒对话框）类是最通用的 Dialog 实现之一。它提供了一些选项让用户对于最通用的对话框使用情形构建屏幕，主要包括以下内容。

1）提供 1～3 个可选按钮来向用户表达信息。这个功能可能和用户在任何桌面编程中的经历相似，显示的按钮一般从 OK、Cancel、Yes 和 No 中选择。

2）以 CheckBox 或 Radio Button 的方式提供选项列表。

3）提供一个供用户输入的文本输入框。例如，创建一个 AlertDialog UI（提醒对话框界面），通过创建一个 AlertDialog.Builder 对象，如下所示。

```
AlertDialog.Builder ad = new AlertDialog.Builder(context);
```

然后，可以设定显示的标题和信息，以可选的形式设定使用的按钮，选择项和用户输入框，还包括设定时间监听来处理用户交互。下面的代码给出了一个新的 AlertDialog 的例子，

用于显示一个信息并提供两个按钮选项供选择，点击任何一个按钮，在执行完附加的 Click Listener（单击监听）后自动关闭。

```
Context context = MyActivity.this;
String title = "It is Pitch Black";
String message = "You are likely to be eaten by a grue.";
String button1String = "Go Back";
String button2String = "Move Forward";
AlertDialog.Builder ad = new AlertDialog.Builder(context);
ad.setTitle(title);
ad.setMessage(message);
ad.setPositiveButton(button1String, new OnClickListener() {
public void onClick(DialogInterface dialog, int arg1) {
eatenByGrue();
}
});
ad.setNegativeButton(button2String, new OnClickListener(){
public void onClick(DialogInterface dialog, int arg1) {
}
});
ad.setCancelable(true);
ad.setOnCancelListener(new OnCancelListener() {
public void onCancel(DialogInterface dialog) {
eatenByGrue();
}
});
```

为了显示已经创建的 AlertDialog，需要调用 show 方法。

```
ad.show();
```

同理，也可以在 Activity 中重写 onCreateDialog（创建对话框）和 onPrepareDialog（准备对话框）方法来创建单实例的对话框来保持它们的状态。这个技巧将在本章的后面进行讲解。

2．特殊的输入Dialog

对话框的一个主要用途是提供用户输入的界面。Android 包含一些特殊的对话框，它们封装了控件为通用的用户输入请求提供了便利。它们包括以下的对话框。

（1）DatePickerDialog

DatePickerDialog 让用户从 DatePicker View 中选择一个日期。构造函数包含一个回调 Listener，用来提示调用的 Activity 日期设定结束。

（2）TimePickerDialog

TimePickerDialog 和 DatePickerDialog 相似，这个对话框让用户从一个 TimePicker View 中选择一个时间。

（3）ProgressDialog

ProgressDialog 是一个在消息文本框下显示了一个进度条的对话框。常用于在一个耗时的操作中，让用户了解当前的进度。



3．使用和管理Dialog

与其在每次需要的时候创建新的对话框实例，不如利用 Android 提供的 onCreateDialog 和 onPrepareDialog 事件处理函数。在 Activity 中，通过这些处理函数来维护和管理对话框实例。

通过重写 onCreateDialog 方法，用户可以设定需要创建的对话框，当调用 showDialog 时，显示指定的对话框。和给出的代码片段一样，重写的方法包含一个 switch 语句来决定哪个对话框需要使用。

```
static final private int TIME_DIALOG = 1;
@Override
public Dialog onCreateDialog(int id) {
switch(id)
{
case (TIME_DIALOG) :
AlertDialog.Builder timeDialog = new AlertDialog.Builder(this);
timeDialog.setTitle("The Current Time Is...");
timeDialog.setMessage("Now");
return timeDialog.create();
}
return null;
}
```

完成初始化的创建后，每次 showDialog 的调用都会触发 onPrepareDialog 处理函数。通过重写这个方法，用户可以在对话框显示之前及时地修改它。在 onPrepareDialog 方法里，用户可以实时地修改任何显示的值，例如，通过下面的代码可以指定当前的时间给上面创建的对话框。

```
@Override
public void onPrepareDialog(int id, Dialog dialog) {
switch(id) {
case (TIME_DIALOG) :
SimpleDateFormat sdf = new SimpleDateFormat("HH:mm:ss");
Date currentTime;
currentTime = new Date(java.lang.System.currentTimeMillis());
String dateString = sdf.format(currentTime);
AlertDialog timeDialog = (AlertDialog)dialog;
timeDialog.setMessage(dateString);
break;
}
}
```

一旦重写了这些方法，就可以通过调用 showDialog 来显示对话框，如下所示。传入想显示的对话框的 ID，Android 会在显示之前创建（如果需要）和准备对话框。

```
showDialog(TIME_DIALOG);
```

作为改善资源利用的方式，这个技巧让用户的 Activity 能处理对话框中的状态信息维持。如何选择或数据输入（例如，项目选择和文本输入）都会在每个对话框实例显示之间维持。

第 5 章　分解Android应用程序核心

在本书前面的章节中，已经讲解了 Android 开发环境的搭建过程和 Android SDK 框架的基本知识，并且通过简单的实例程序，分解了各段代码。在本章内容中，将进一步分解 Android 应用程序，详细剖析 Android 应用程序的核心构成部分，为读者学习本书后面的知识打下基础。

5.1　Android体系结构介绍

Android 作为一个移动设备平台，其软件层次结构包括了一个操作系统（OS），中间件（MiddleWare）和应用程序（Application）。根据 Android 的软件框图，其软件层次结构自下而上分为以下几个层次。

1）操作系统层（OS）。

2）各种库（Libraries）和 Android 运行环境（RunTime）。

3）应用程序框架（Application Framework）。

4）应用程序（Application）。

具体结构如图 5-1 所示。

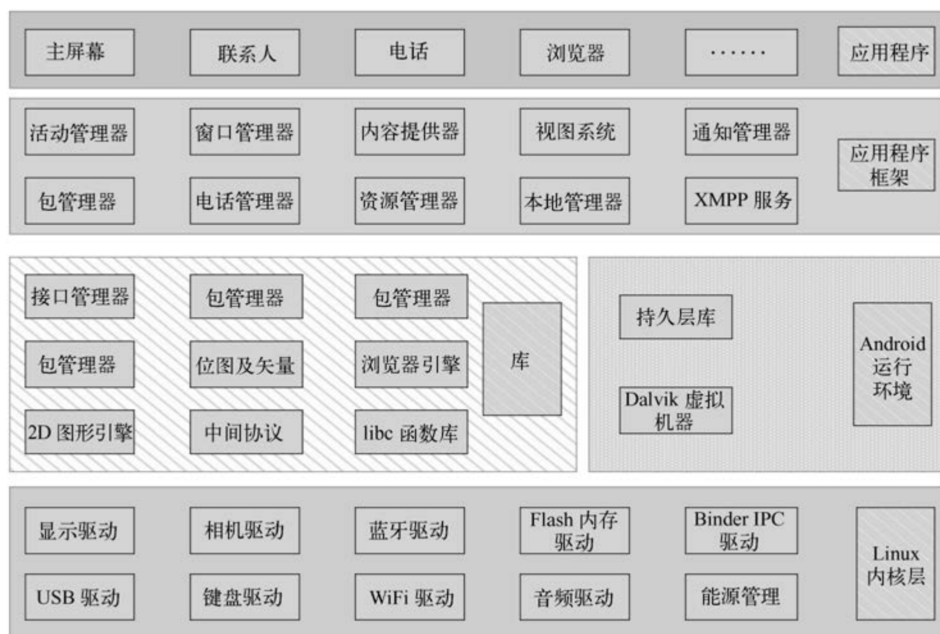


图 5-1　Android 操作系统的组件结构图



下面详细介绍 Andoid 各个层次的软件的重点及其相关技术。

5.1.1　操作系统层

Android 对操作系统的使用包括核心和驱动程序两部分，Android 的 Linux 核心为标准的 Linux 2.6 内核，Android 更多的是需要一些与移动设备相关的驱动程序。主要的驱动如下所示。

- ❑ 显示驱动（Display Driver）：常用基于 Linux 的帧缓冲（Frame Buffer）驱动；
- ❑ Flash 内存驱动（Flash Memory Driver）。是基于 MTD 的 Flash 驱动程序。
- ❑ 照相机驱动（Camera Driver）：常用基于 Linux 的 v4l（Video for ）驱动。
- ❑ 音频驱动（Audio Driver）：常用基于 ALSA（Advanced Linux Sound Architecture，高级 Linux 声音体系）驱动。
- ❑ Wi-Fi 驱动（Camera Driver）：基于 IEEE 802.11 标准的驱动程序。
- ❑ 键盘驱动（KeyBoard Driver）。作为输入设备的键盘驱动。
- ❑ 蓝牙驱动（Bluetooth Driver）。基于 IEEE 802.15.1 标准的无线传输技术。
- ❑ Binder IPC 驱动：Andoid 一个特殊的驱动程序，具有单独的设备节点，提供进程间通信的功能。
- ❑ Power Management（能源管理）：管理电池电量等信息。

5.1.2　各种库和Android 运行环境

本层次对应一般嵌入式系统，相当于中间件层次。本层次分成两个部分，一个是各种库，另一个是 Android 运行环境。本层次的内容大多是使用 C++实现的。其中包含的各种库如下。

- ❑ C 库：C 语言的标准库，也是系统中一个最为底层的库，C 库是通过 Linux 的系统调用来实现。
- ❑ 多媒体框架（MediaFrameword）：这部分内容是 Android 多媒体的核心部分，基于 PacketVideo（即 PV）的 OpenCORE，从功能上本库一共分为两大部分，一个部分是音频、视频的回放（PlayBack）；另一部分是则是音频、视频的纪录（Recorder）。
- ❑ SGL：2D 图像引擎。
- ❑ SSL：即 Secure Socket Layer 位于 TCP/IP 协议与各种应用层协议之间,为数据通信提供安全支持。
- ❑ OpenGL ES 1.0：提供了对 3D 的支持。
- ❑ 界面管理工具（Surface Management）：提供了管理显示子系统等功能。
- ❑ SQLite：一个通用的嵌入式数据库。
- ❑ WebKit：网络浏览器的核心。
- ❑ FreeType：位图和矢量字体的功能。

Android 的各种库一般是以系统中间件的形式提供的，它们都有的一个显著特点就是与移动设备的平台的应用密切相关。

Android 运行环境主要指的是虚拟机技术——Dalvik。Dalvik 虚拟机和一般 Java 虚拟机（Java VM）不同，它执行的不是 Java 标准的字节码（Bytecode）而是 Dalvik 可执行格式（.dex）中的执行文件。在执行的过程中，每一个应用程序即一个进程。二者最大的区别在于 Java VM 是基于栈的虚拟机（Stack-based），而 Dalvik 是基于寄存器的虚拟机（Register-based）。显然，

后者最大的好处在于可以根据硬件实现更大的优化，这更适合移动设备。

5.1.3 应用程序

Android 的应用程序主要是用户界面（User Interface）方面的，通常以 Java 程序编写，其中还可以包含各种资源文件（放置在 res 目录中）。Java 程序及相关资源经过编译后，将生成一个 APK 包。Android 本身提供了主屏幕（Home）、联系人（Contact）、电话（Phone）、浏览器（Browers）等众多的核心应用。而应用程序的开发者则可以使用应用程序框架层的 API 实现自己的程序，这也是 Android 开源的巨大潜力的体现。

5.1.4 应用程序框架

Android 的应用程序框架为应用程序层的开发者提供 API。由于上层的应用程序是以 Java 构建的，因此本层次提供的首先包含了 UI 程序中所需要的各种控件，例如，Views（视图组件），其中又包括了 List（列表）、Grid（栅格）、Text Box（文本框）、Button（按钮）等，甚至一个嵌入式的 Web 浏览器。

5.2 Android应用程序组成

一个 Android 应用程序通常有下面 5 个组件组成，分别是 Activity、Intent Receiver、Service、Content Provider、Intent and Intent Filters。在本节的内容中，将详细介绍这 5 类组件的基本知识。

5.2.1 Activity介绍

Activity 是 5 个组件中最常用的，程序中 Activity 通常的表现形式是一个单独的界面（Screen）。每个 Activity 都是一个单独的类，它扩展实现了 Activity 基础类。这个类显示为一个由 Views 组成的用户界面，并响应事件。大多数程序有多个 Activity。例如，一个文本信息程序有这样几个界面：显示联系人列表界面、写信息界面、查看信息界面或者设置界面等。每个界面都是一个 Activity。切换到另一个界面就是载入一个新的 Activity。某些情况下，一个 Activity 可能会给前一个 Activity 返回值。例如，一个让用户选择相片的 Activity 会把选择到的相片返回给其调用者。

打开一个新界面后，前一个界面就被暂停，并放入历史栈中（界面切换历史栈）。使用者可以回溯前面已经打开的存放在历史栈中的界面，也可以从历史栈中删除没有价值的界面。Android 在历史栈中保留程序运行产生的所有界面：从第一个界面到最后一个。

5.2.2 Broadcast Intent Receiver介绍

当想要执行一些与外部事件相关的代码时，如当来电响铃或数据网络可用时，执行某些命令。用户就需要使用 Broadcast Intent Receiver 了。Broadcast Intent Receivers 没有 UI，尽管它们使用 Notification Manager 来通知用户发现了某些事件动作。Broadcast Intent Receivers 在 AndroidManifest.xml 文件中声明，不过用户可以使用 Context.registerReceiver()来声明。用户的程序没有必要运行来等待 Broadcast Intent Receivers 被调用。当一个 Broadcast Intent Receiver



被触发时，如果需要的话，系统自然会启动用户的程序，程序也可以通过 Context.broadcastIntent()来发送自己的 Intent 广播给其他程序。

5.2.3　Service（服务）介绍

Service 是一个没有 UI（用户界面）且长驻系统的代码。最佳例子是媒体播放器从播放列表中播放歌曲。媒体播放器程序中，可能有一个或多个 Activity 让用户选择歌曲播放。然而，在后台播放歌曲就无需 Activity 干涉了，因为用户希望在音乐播放的同时能够切换到其他界面。此时，媒体播放器 Activity 需要通过 Context.startService()启动一个 Service，这个 Service 在后台运行以保持继续播放音乐。在媒体播放器被关闭之前，系统会保持音乐后台播放 Service 的正常运行。我们可以用 Context.bindService()方法连接到一个 Service 上，如果 Service 未运行的话，连接后，也会启动它，连接上之后就可以通过一个 Service 提供的接口与 Service 进行通话，例如对音乐播放 Service 来说，则提供了暂停，重放等功能。

1．如何使用Service

在 Android 中有如下两种方法使用 Service。

1）通过调用 Context.startService()启动，调用 Context.stopService()结束，Context.startService()可以传递参数给 Service。

2）通过调用 Context.bindService()启动，调用 Context.unbindService()结束，还可以通过 ServiceConnection 访问 Service。

上述二种方式可以混合使用，比如说可以先使用 startSerice()再使用 unbindService()。

2．Service的生命周期

在 startService()后，即使调用 startService()的进程结束了，Service 仍然还存在，知道有进程调用 stoptService()，或者 Service 调用 StopSelf()方法自己停止。

在 bindService()绑定服务后，Service 就和调用 bindService()的进程同步，也就是说当调用 bindService()的进程结束了，那么它 bind 的 Service 也要跟着被结束，当然期间也可以调用 unbindService()让 Service 结束。

当混合使用这两种方式时，例如，甲 startService()了，乙 bindService()了，那么只有甲 stoptService()了而且乙也 unbindService()时，这个 Service 才会被结束。

3．进程生命周期

Android 系统将会尝试保留那些启动了的或者是绑定了的服务进程。如果该服务正在执行 onCreate(),onStart()或者 onDestroy()这些方法，那么主进程将会成为一个前台进程，以确保此代码不会被停止。

如果服务已经开始，那么它的主进程就重要性而言低于所有可见的进程但高于不可见的进程，由于只有少数几个进程是用户可见的，所以只要不是内存特别小，该服务不会停止。

如果有多个客户端绑定了服务，只要客户端中的一个对于用户是可见的，即认为该服务可见。

5.2.4　Content Provider介绍

应用程序把数据存放在一个 SQLite 数据库格式文件里，或者存放在其他有效设备里。如果用户想让其他程序能够使用自己程序的数据，Content Provider 就很有用了。Content Provider

是一个实现了一系列标准方法的类，这个类使得其他程序能存储、读取某种 Content Provider 可处理的数据。

5.2.5 Intent和Intent Filter

Android 通过一个专门的 Intent 类来进行界面的切换，Intent 类描述了程序的意图。数据结构有两个最重要的部分，分别是操作（Action）与按照既定规则处理的数据（Data）。典型的操作包括 Main (Activity 的入口)、View、Pick、Edit 等等，数据用 URI 表示。例如，查看某人的联系信息，需要创建一个 Intent，使用 View 操作，数据则是一个指向此人的 URI。

有个相关的类叫 Intent Filter。Intent 类是一个请求来做什么事情；而 Intent Filter 类则描述了一个 Activity（或下文的 Intent Receiver）能处理什么意图。显示某人联系信息的 Activity 使用了一个 Intent Filter，就是说它知道如何处理应用到此人数据的 View 操作。Activity 在 AndroidManifest.xml 文件中使用 Intent Filter。

通过解析 Intents 来完成 Activity 的切换。使用 startActivity(myIntent)来启用新的 Activity。系统考察所有安装程序的 Intent Filters，然后找到与 myIntent 匹配最好的 Intent Filters 所对应的 Activity。这个新 Activity 接到 Intent 传来的消息，并因此被启用。解析 Intents 的过程发生在 startActivity 被实时调用时，这样做有以下 2 个好处。

1）Activities 仅发出一个 Intent 请求，便能重用其他组件的功能。

2）Activities 可以随时被替换为有等价 Intent Filter 的新 Activity。

注意：Intent 和 Intent Filter 是不常用的构成部分，通常被很多教材排除在应用程序组成之外。

5.3 Android应用工程文件组成

Android 的应用工程文件主要由以下部分组成。

- src 文件：源文件都在这个目录里面。
- R.java 文件：这个文件是 Eclipse 自动生成的，应用开发者不需要去修改里边的内容。
- Android Library：这个是应用运行的 Android 库。
- assets 目录：里面主要放置多媒体等一些文件。
- res 目录：里面主要放置应用用到的资源文件。
- drawable 目录：主要放置应用到的图片资源。
- layout 目录：主要放置用到的布局文件。这些布局文件都是 XML 文件。
- values 目录：主要放置字符串（strings.xml）、颜色（colors.xml）、数组（arrays.xml）。
- Androidmanifest.xml：相当于应用的配置文件。在这个文件里边，必须声明应用的名称，应用所用到的 Activity，Service，以及 Receiver 等。

在 Eclipse 中，一个基本的 Android 项目的目录结构如图 5-2 所示。



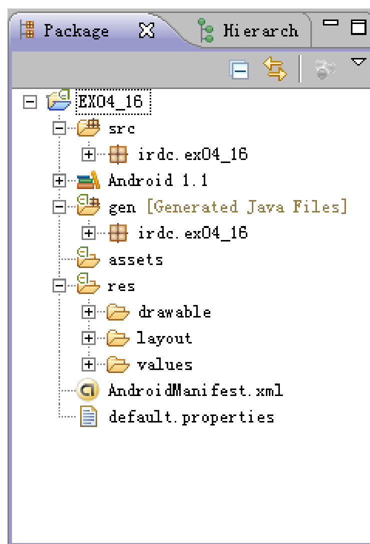


图 5-2　Android 应用工程文件组成

5.3.1　src目录

与一般的 Java 项目一样，src 文件夹是项目的所有包及源文件（.java），res 文件夹中则包含了项目中的所有资源，例如，程序图标（drawable）、布局文件（layout）和常量（values）等。不同的是，在 Java 项目中没有 gen 文件夹，也没有每个 Android 项目都必须有的 AndroidManfest.xml 文件。

.java 格式文件是在建立项目时自动生成的，这个文件是只读模式，不能更改，R.java 文件是定义该项目所有资源的索引文件。先来看看 HelloAndroid 项目的 R.java 文件，代码如下。

```
package com.yarin.Android.HelloAndroid;
public final class R {
    public static final class attr {
    }
    public static final class drawable {
        public static final int icon=0x7f020000;
    }
    public static final class layout {
        public static final int main=0x7f030000;
    }
    public static final class string {
        public static final int app_name=0x7f040001;
        public static final int hello=0x7f040000;
    }
}
```

从上述代码中，可以看到定义了很多常量，还会发现这些常量的名字都与 res 文件夹中的文件名相同，这再次证明.java 文件中所存储的是该项目所有资源的索引。有了这个文件，在程序中使用资源将变得更加方便，可以很快地找到要使用的资源，由于这个文件不能被手动

编辑，所以当用户在项目中加入了新的资源时，只需要刷新一下该项目，.java 文件便自动生成了所有资源的索引。

5.3.2 AndroidManfest.xml文件

AndroidManfest.xml 文件包含了该项目中所使用的 Activity、Service、Receiver，用户先来打开 HelloAndroid 项目中的 AndroidManfest.xml 文件，代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.yarin.Android.HelloAndroid"
        android:versionCode="1"
        android:versionName="1.0">
    <application android:icon="@drawable/icon"
android:label="@string/app_name">
        <activity android:name=".HelloAndroid"
                android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
    <uses-sdk android:minSdkVersion="5" />
</manifest>
```

在上述代码中，Intent Filter 描述了 Activity 启动的位置和时间。每当一个 Activity（或者操作系统）要执行一个操作时，它将创建出一个 Intent 对象，这个 Intent 对象能承载的信息可描述用户想做什么，用户想处理什么数据，数据的类型，以及一些其他信息。而 Android 则会和每个 Application 所暴露的 Intent Filter 的数据进行比较，找到最合适的 Activity 来处理调用者所指定的数据和操作。下面仔细分析 AndroidManfest.xml 文件，如表 5-1 所示。

表 5-1　AndroidManfest.xml 分析

参　　数	说　　明
manifest	根节点，描述了 package 中所有的内容
xmlns:android	包含命名空间的声明。xmlns:android=http://schemas.android.com/apk/res/android，使得 Android 中各种标准属性能在文件中使用，提供了大部分元素中的数据
Package	声明应用程序包
application	包含 package 中 application 级别组件声明的根节点。此元素也可包含 application 的一些全局和默认的属性，如标签、icon、主题、必要的权限，等等。一个 manifest 能包含零个或一个此元素（不能大于一个）
android:icon	应用程序图标
android:label	应用程序名字
Activity	用来与用户交互的主要工具。Activity 是用户打开一个应用程序的初始页面，大部分被使用到的其他页面也由不同的 Activity 所实现，并声明在另外的 Activity 标记中。注意，每一个 Activity 必须有一个<activity>标记对应，无论它给外部使用还是只用于自己的 package 中。如果一个 Activity 没有对应的标记，用户将不能运行它。另外，为了支持运行时查找 Activity，可包含一个或多个<intent-filter>元素来描述 Activity 所支持的操作



（续）

参　　数	说　　明
android:name	应用程序默认启动的 Activity
intent-filter	声明了指定的一组组件支持的 Intent 值，从而形成了 Intent Filter。除了能在此元素下指定不同类型的值，属性也能放在这里来描述一个操作所需的唯一的标签、icon 和其他信息
action	组件支持的 Intent action
category	组件支持的 Intent Category。这里指定了应用程序默认启动的 Activity
uses-sdk	该应用程序所使用的 SDK 版本相关

5.3.3　常量的定义文件

下面介绍资源文件中一些常量的定义，如 String.xml，代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, HelloAndroid!</string>
    <string name="app_name">HelloAndroid</string>
</resources>
```

上述代码很简单，就定义了两个字符串资源。

接下来分析 HelloAndroid 项目的布局文件（layout），首先打开 res->layout->main.xml 文件，对应代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

在上述代码中，有以下几个布局和参数。

- < LinearLayout></LinearLayout>：线性版面配置，在这个标签中，所有元件都是按由上到下的顺序排成的。
- android:orientation：表示这个介质的版面配置方式是从上到下垂直地排列其内部的视图。
- android:layout_width：定义当前视图在屏幕上所占的宽度，fill_parent 即填充整个屏幕。
- android:layout_height：定义当前视图在屏幕上所占的高度，fill_parent 即填充整个屏幕。
- wrap_content：随着文字栏位的不同而改变这个视图的宽度或高度。

在上述布局代码中，使用了 1 个 TextView 来配置文本标签 Widget（构件）。其中设置的

属性 android:layout_width 为整个屏幕的宽度，android:layout_height 可以根据文字来改变高度，而 android:text 则设置了这个 TextView 要显示的文字内容，这里引用了@string 中的 hello 字符串，即 String.xml 文件中的 hello 所代表的字符串资源。hello 字符串的内容“Hello World, HelloAndroid!”这就是我们在 HelloAndroid 项目运行时看到的字符串。

注意：上面介绍的文件只是主要文件，在项目中需要用户自行编写。在项目中还有很多其他的文件，那些文件很少需要用户编写的，所以在此就不进行讲解了。

5.4 应用程序的生命周期

程序也如同自然界的生物一样，有自己的生命周期。应用程序的生命周期即程序的存活时间，即在什么时间内有效。Android 是一构建在 Linux 之上的开源移动开发平台，在 Android 中，多数情况下每个程序都是在各自独立的 Linux 进程中运行的。当一个程序或其某些部分被请求时，它的进程就“出生”了；当这个程序没有必要再运行下去且系统需要回收这个进程的内存用于其他程序时，这个进程就“死亡”了。可以看出，Android 程序的生命周期是由系统控制而非程序自身直接控制的。这和编写桌面应用程序时的思维有一些不同，一个桌面应用程序的进程也是在其他进程或用户请求时被创建，但往往是在程序自身收到关闭请求后执行一个特定的动作（比如从 main 函数中 return）而导致进程结束的。要想做好某种类型的程序或者某种平台下的程序的开发，最关键的就是要弄清楚这种类型的程序或整个平台下的程序的一般工作模式并熟记在心。在 Android 中，程序的生命周期控制就是属于这个范畴。

开发者必须理解不同的应用程序组件（尤其是 Activity、Service 和 Intent Receiver）是如何影响应用程序的生命周期的，错误地使用这些组件可能会导致系统终止正在执行重要任务的应用程序进程。

一个常见的进程生命周期出错的例子是 Intent Receiver（意图接收器）的。当 Intent Receiver 在 onReceive()方法中接收到一个 Intent（意图）时，它会启动一个线程，然后返回。一旦返回，系统将认为 Intent Receiver 不再处于活动状态，因而 Intent Receiver 所在的进程也就不再有用了（除非该进程中还有其他的组件处于活动状态）。因此，系统可能会在任意时刻终止该进程以回收占有的内存，这时进程中创建出的那个线程也将被终止。解决这个问题的方法是从 Intent Receiver 中启动一个服务，让系统知道进程中还有处于活动状态的工作。为了使系统能够正确决定在内存不足时应该终止哪个进程，Android 根据每个进程中运行的组件及组件的状态把进程放入一个“Importance Hierarchy（重要性分级）”中。进程的类型按重要程度排序如下。

1．前台进程（Foreground）

前台进程与用户当前正在做的事情密切相关。不同的应用程序组件能够通过不同的方法将它的宿主进程移到前台。在如下的任何一个条件下：进程正在屏幕的最前端运行一个与用户交互的活动（Activity），它的 onResume()方法被调用；或进程有一正在运行的 Intent Receiver（它的 IntentReceiver.onReceive()方法正在执行）；或进程有一个服务（Service），并且在服务的某个回调函数（Service.onCreate()、 Service.onStart()或 Service.onDestroy()）内有正在执行的代码，系统将把进程移动到前台。



2．可见进程（Visible）

可见进程有一个可以被用户从屏幕上看到的活动（Activity），但不在前台（它的 onPause()方法被调用）。如果前台的活动是一个对话框，以前的活动就隐藏在对话框之后，就出现这种进程。进程非常重要，一般不允许被终止，除非是为了保证前台进程的运行而不得不终止它。

3．服务进程（Service）

服务进程有一个已经用 startService()方法启动的服务。虽然用户无法直接看到这些进程，但它们做的事情却是用户所关心的（如后台 MP3 回放或后台网络数据的上传下载）。因此，系统将一直运行这些进程，除非内存不足以维持所有的前台进程和可见进程。

4．后台进程（Background）

后台进程有一个当前用户看不到的活动（它的 onStop()方法被调用）。这些进程对用户体验没有直接的影响。如果它们正确执行了活动生命周期，系统可以在任意时刻终止该进程以回收内存，并提供给前面 3 种类型的进程使用。系统中通常有很多这样的进程在运行，因此要将这些进程保存在 LRU 列表中，以确保当内存不足时用户最近看到的进程最后一个被终止。

5．空进程（Empty）

空进程不拥有任何活动的应用程序组件的进程。保留这种进程的唯一原因是在下次应用程序的某个组件需要运行时，不需要重新创建进程，这样可以提高启动速度。

系统将以进程中当前处于活动状态组件的重要程度为基础对进程进行分类。进程的优先级可能也会根据该进程与其他进程的依赖关系而增长。例如，如果进程 A 通过在进程 B 中设置 Context.BIND_AUTO_CREATE 标记或使用 ContentProvider 被绑定到一个服务（Service），那么进程 B 在分类时至少要被看成与进程 A 同等重要。

5.5　Activity的生命周期

在 Android 中，一般用系统管理来决定进程的生命周期。有时因为手机所具有的一些特殊性，所以我们需要更多地去关注各个 Android 程序部分的运行时生命周期模型。所谓手机的特殊性，主要是指如下两点。

1）在进行手机应用时，大多数情况下只能在手机上看到一个程序的一个界面，用户除了通过程序界面上的功能按钮在不同的窗体间切换，还可以通过 Back（返回）键和 Home（主）键来返回上一个窗口，而用户使用 Back 键或者 Home 键的时机是非常不确定的，任何时候用户都可以使用 Home 键或 Back 键来强行切换当前的界面。

2）通常手机上一些特殊的事件发生也会强制改变当前用户所处的操作状态，例如，无论任何情况，在手机来电时，系统都会优先显示电话接听界面。

了解了手机应用的上述特殊性之后，接下来将详细介绍 Activity 在不同阶段的生命周期。

5.5.1　Activity的几种状态

要想了解 Activity 在不同阶段的生命周期，首先需要了解 Activity 的几种状态。当 Activity 被创建或销毁时，会存在如下 4 种状态。

- Active（活动）：当 Activity 在栈的顶端时，它是可见的，有焦点的前台 Activity，用来响应用户的输入。Android 会不惜一切代价来尝试保证它的活跃性，需要的话它会

杀死栈中更靠下的 Activity 来保证 Active Activity 需要的资源。当另一个 Active 变成活动状态时，这个就会暂停。

- Paused（暂停）：在一些情况下， Activity 可见但不拥有焦点；在这个时刻，它就是暂停的。当最前面的 Activity 是全透明或非全屏的 Activity 时，下面的 Activity 就会到达这个状态。当暂停时，这个 Activity 还是被看做是活动的，但不接受用户的输入事件。在极端的情况下，Android 会杀死一个暂停的 Activity 来恢复资源给 Active Activity。当一个 Activity 完全不可见时，它就变成停止。
- Stopped（停止）：当一个 Activity 不可见，它就“停止”了。这个 Activity 仍然留在内存里来保存所有的状态和成员信息；但是，当系统需要内存时，它就会被处理掉。当一个 Activity 停止时，保存数据和当前 UI 状态是很重要的。一旦 Activity 退出或关闭，它就被销毁了。
- Inactive（销毁）：当一个曾经被启动过的 Activity 被杀死时，它就被销毁了。Inactive Activity 会从 Activity 栈中移除，当它重新显示和使用时需要再次启动。

Activity 状态转换如图 5-3 所示。

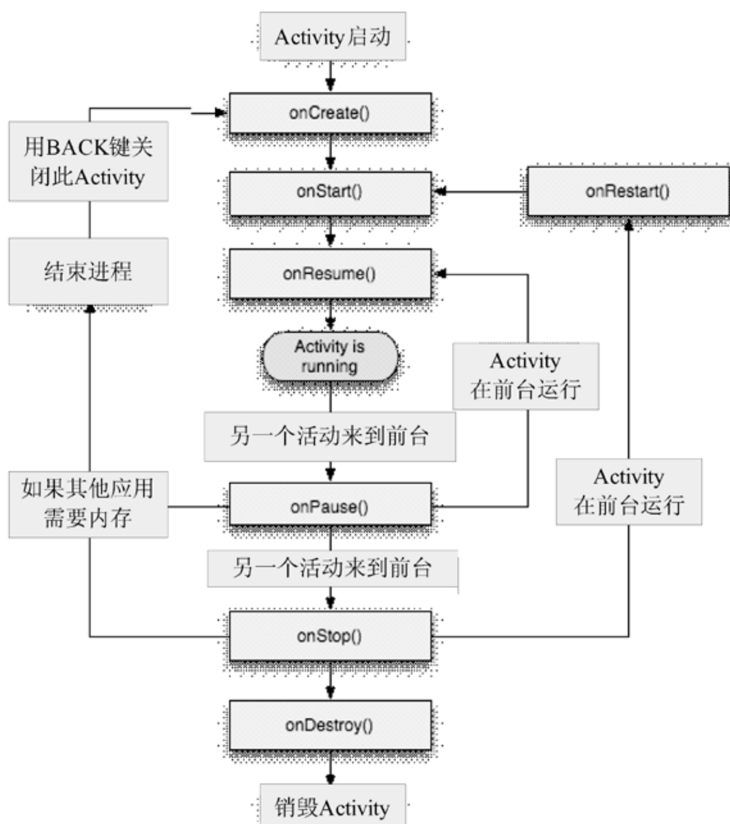


图 5-3 Activity 状态转换图

图 5-3 所示的状态的变化是由 Android 内存管理器决定的，Android 会首先关闭那些包含 Inactive Activity 的应用程序，然后关闭 Stopped 状态的程序。在极端情况下，会移除 Paused 状态的程序。



5.5.2　分解剖析Activity

1. void onCreate(Bundle savedInstanceState)

当 Activity 被第一次加载时执行 onCreate()事件，当新启动一个程序的时候，其主窗体的 onCreate 事件就会被执行。如果 Activity 被 onDestroy（销毁）后，再重新加载 Task（任务）时，其 onCreate()事件也会被重新执行。

2. void onStart()

在 onCreate()事件之后执行 onStart()事件。或者当前窗体被交换到后台后，在用户重新查看窗体前已经过去了一段时间，窗体已经执行了 onStop()事件，但是窗体和其所在进程并没有被销毁，用户再次重新查看窗体时会执行 onRestart()事件，之后会跳过 onCreate()事件，直接执行窗体的 onStart 事件。

3. void onResume()

onStart()事件之后执行 onResume()事件。或者当前窗体被交换到后台后，在用户重新查看窗体时，窗体还没有被销毁，也没有执行过 onStop()事件（窗体还继续存在于 Task 中），则会跳过窗体的 onCreate()和 onStart()事件，直接执行 onResume()事件。

4. void onPause()

窗体被交换到后台时执行 onPause()事件。

5. void onStop()

onPause()事件之后执行 onStop()。如果一段时间内用户还没有重新查看该窗体，则该窗体的 onStop()事件将会被执行；或者用户直接按了 Back 键，将该窗体从当前 Task 中移除，也会执行该窗体的 onStop()事件。

6. void onRestart()

onStop()事件执行后执行 onRestart()，如果窗体和其所在的进程没有被系统销毁，此时用户又重新查看该窗体，则会执行窗体的 onRestart 事件，onRestart 事件后会跳过窗体的 onCreate()事件直接执行 onStart()事件。

7. void onDestroy()

Activity 被销毁的时候执行 onDestroy()事件。在窗体的 onStop()事件之后，如果没有再次查看该窗体，Activity 则会被销毁。

5.6　Android进程与线程

当某个组件第一次运行的时候，Android 启动了一个进程。默认的，所有的组件和程序运行在这个进程和线程中。也可以安排组件在其他的进程或者线程中运行。在本节的内容中，将简要介绍 Android 进程与线程的基本知识。

5.6.1　进程

组件运行的进程由 Manifest File 控制，组件中的节点<activity>、<service>、<receiver>和<provider>都包含了一个 Process 属性。通过 Process 属性，可以设置组件运行的进程，既可以配置组件在一个独立进程中运行，也可以配置多个组件在同一个进程运行，甚至可以配置多

个程序在同一个进程中运行（前提是这些程序共享一个 User ID 并给定同样的权限）。另外，在<application> 节点中也包含了 Process 属性，可以用来设置程序中所有组件的默认进程。

所有的组件在此进程的主线程中实例化，系统对这些组件的调用从主线程中分离。并非每个对象都会从主线程中分离。一般来说，响应如 View.onKeyDown()用户操作的方法和通知的方法也在主线程中运行。这就表示，组件被系统调用的时候不应该长时间运行或者阻塞操作（如网络操作或者计算大量数据），因为这样会阻塞进程中的其他组件。可以把这类操作从主线程中分离。

当更加常用的进程无法获取足够内存，Android 可能会关闭不常用的进程。下次启动程序的时候会重新启动进程。

当决定哪个进程需要被关闭的时候，Android 会考虑哪个对用户更加有用。如 Android 会倾向于关闭一个长期不显示在界面的进程来支持一个经常显示在界面的进程。是否关闭一个进程取决于组件在进程中的状态，参见后面的章节 Component Lifecycles。

5.6.2 线程

即使为组件分配了不同的进程，有时候也需要再分配线程。比如用户界面需要很快对用户进行响应，因此某些费时的操作，如网络连接、下载或者非常占用服务器时间的操作应该放到其他线程。

线程通过 Java 的标准对象 Thread 创建，在 Android 中提供了如下管理线程的方法。

- Looper：在线程中运行一个消息循环。
- Handler：传递一个消息。
- HandlerThread：创建一个带有消息循环的线程。

Android 会让一个应用程序在单独的线程中，指导它创建自己的线程。除了上述方法外，通过使用应用程序组件，如 Activity、service、broadcast receiver，可以在主线程中实现实例化操作。

5.6.3 线程安全的方法

了解了进程和线程的基本知识后，开发人员很有必要了解线程安全方面的知识。在某些情况下，程序中的方法可能不止调用一个线程，这样在多个线程协同交互工作时，需要特别注意线程安全的问题。例如有一个方法正在调用在 IBinder 接口对象，此对象中的方法的程序启动了和 IBinder 对象相同的进程，这个方法就相当于在 IBinder 的进程中执行。但是，如果此时调用者发起另外一个进程，这个方法需要在另外一个线程中运行。因为这个线程和 IBinder 对象在同一个线程池中，所以它不会在进程的主线程中运行。假如应用中的一个 Service（服务）从主线程被调用 onBind() 方法，onBind() 返回的对象中的方法会被从线程池中调用。这样就造成了一个服务被多个客户端请求的情景，不止一个线程池会在同一时间调用 IBinder 中的方法，所以此时 IBinder 必须保证线程安全。如果不安全，则会影响多个线程，从而影响和 IBinder 相关的所有应用。

第 6 章　Android常用组件

组件是编程中的重要组成部分，一个程序通常由多个组件共同构成以实现某项具体功能。在 Android SDK 中，可以通过大量的组件来实现具体项目的需求。本章将详细介绍 Android 中基本组件的知识，并通过具体实例的实现过程讲解各个组件的使用方法，为读者学习本书后面的知识打下坚实的基础。

6.1　UI（界面）组件

一个 Android 程序是由一个或多个 Activity（活动）组成的，Activity 是一个 UI（用户界面）容器，它本身不在用户界面中显示。在本节的内容中，将首先简单讲解编程中用到的 UI 基本组件的基本知识。

6.1.1　视图组件——View

View 是一个最基本的 UI 类，几乎所有的 UI 组件都是继承 View 而实现的。例如，本书第 4 章中的例子，就用到了 TextView，其使用格式如下。

```
android.view.View
```

其主要功能如下。

- 为指定的屏幕矩形区域存储布局和内容。
- 处理尺寸和布局，绘制，焦点改变，翻屏，按键、手势。

Android 中的常用 View 类如表 6-1 所示。

表 6-1　常用 View 类

文本 TextView	输入框 EditText
输入法 InputMethod	活动方法 MovementMethod
按钮 Button	单选按钮 RadioButton
复选框 Checkbox	滚动视图 ScrollView

6.1.2　视图容器组件——Viewgroup

视图容器组件 Viewgroup 的使用格式如下。

```
android.view.Viewgroup
```

Viewgroup 用于包含并管理下级系列的 Views 和其他 Viewgroup，是一个布局的基类。

ViewGroup 好像一个 View 容器，负责对添加进来的 View 进行布局处理。一个 ViewGroup 可以建到另一个 ViewGroup 中去。这是因为 ViewGroup 也是继承于 View.ViewGroup 类，即其他容器类的基类。它们之间的关系如图 6-1 所示。

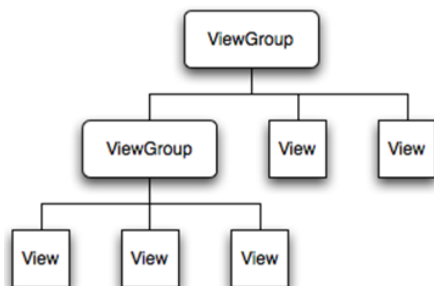


图 6-1　各类的继承关系

6.1.3　布局组件——Layout

布局就像容器，里面可以装下很多控件。布局里面还可以套用其他的布局。这样就可以实现界面的多样化以及设计的灵活性。布局组件 Layout 的使用格式如下。

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
>
```

一个布局容器里可以包括 0 或多个布局容器。LinearLayout 线性的布局方式，要么上下，要么左右地添加控件，常用的 Layout 实现类有如下 5 个。

1）AbsoluteLayout：可以让子元素指定准确的 x/y 坐标值，并显示在屏幕上。(0, 0)为左上角，当向下或向右移动时，坐标值将变大。AbsoluteLayout 没有页边框，允许元素之间互相重叠。通常不推荐使用 AbsoluteLayout，除非有充足理由要使用它，因为它使界面代码太过刚性，以至于可能在不同的设备上不能很好地工作，效果如图 6-2 所示。

2）TableLayout：用于把子元素放入到行与列中，不显示行、列或是单元格边界线，但是单元格不能横跨行，如在 HTML 中一样，效果如图 6-3 所示。

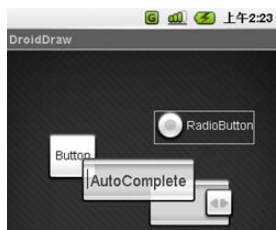


图 6-2　AbsoluteLayout 效果



图 6-3　TableLayout 效果

3）RelativeLayout：允许子元素指定它们相对于其他元素或父元素的位置（通过 ID 指定）。因此，可以以右对齐，或上下，或置于屏幕中央的形式来排列两个元素。元素按顺序排列，



因此如果第一个元素在屏幕的中央，那么相对于这个元素的其他元素将以屏幕中央的相对位置来排列。如果使用 XML 来指定这个 layout，在用户定义它之前，被关联的元素必须定义，其效果结构如图 6-4 所示。



图 6-4　RelativeLayout 效果结构

4）FrameLayout：这是最简单的一个布局对象。它被定制为屏幕上的一个空白备用区域，之后可以在其中填充一个单一对象，比如，一张要发布的图片。所有的子元素将会固定在屏幕的左上角；不能为 FrameLayout 中的一个子元素指定一个位置。后一个子元素将会直接在前一个子元素之上进行覆盖填充，把它们部分或全部挡住（除非后一个子元素是透明的）。

5）LinearLayout：用于在一个方向上（垂直或水平）对齐所有子元素。所有子元素可以一个跟一个地堆放；也可以一个垂直列表每行将只有一个子元素（无论它们有多宽），如图 6-5 所示；也可以一个水平列表只是一列的高度（最高子元素的高度来填充），如图 6-6 所示。

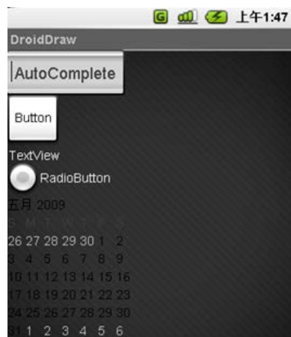


图 6-5　垂直布局

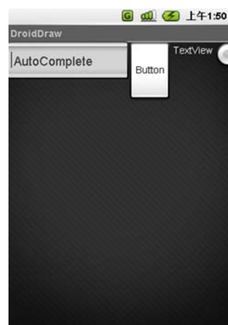


图 6-6　水平布局

6.1.4　布局参数——LayoutParams

当将一个 View 加入到一个 ViewGroup 中后，例如，加入到了 RelativeLayout 里面。此时这个 View 在 RelativeLayout 里面是怎样显示的呢？是显示在左边还是右边？上边还是下边？具体做法是：在向其中加入 View 时，可以传递一组值，并将这组值封装在 LayoutParams 类中。这样当在显示这个 View 时，其容器会根据封装在 LayoutParams 的值来确认此 View 的显示大小和位置。由此可以看出，LayoutParams 的功能如下。

- 每一个 Viewgroup 类使用一个继承于 ViewGroup.LayoutParams 的嵌套类。
- 包含定义了子节点 View 的尺寸和位置的属性类型。

LayoutParams 的具体结构图如图 6-7 所示。

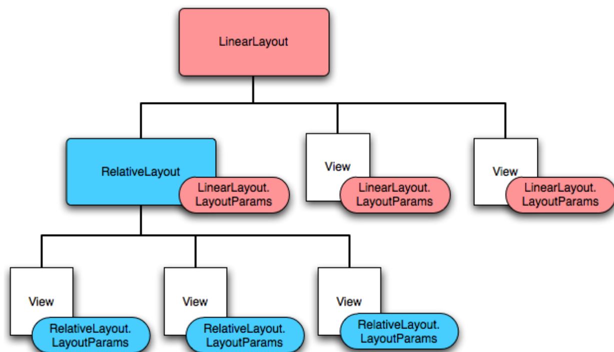


图 6-7　LayoutParams 结构图

6.2　绚丽多彩——应用界面布局实例

在本节的内容中，将通过一个具体实例的实现过程，来讲解使用基本布局控件的方法。本节的所有实例（除 6.2.1 外），对应的源代码文件保存在“光盘\daima\6\Widget”文件夹内。

6.2.1　编程实现

本实例保存在“光盘:\daima\6\”文件夹内，命名为“buju”。本实例的具体实现流程如下所示。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“buju”的工程文件，如图 6-8 所示。

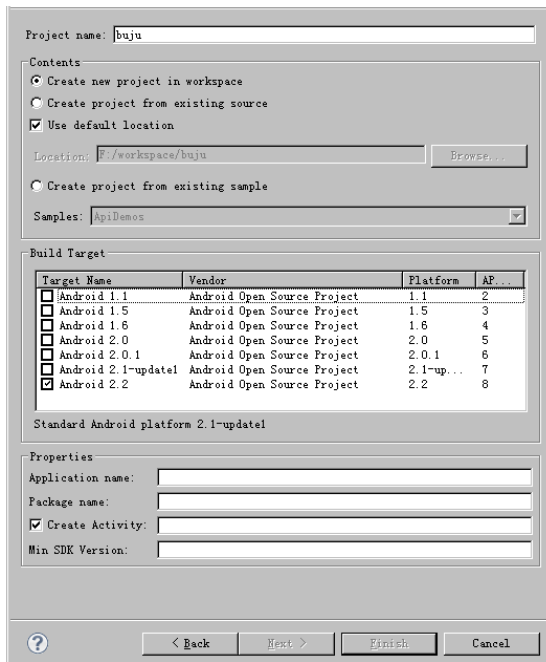


图 6-8　新建一个 Android Project



第 2 步：编写主文件。系统主文件 src\com\eoeandroid\layout\ActivityMain.java.java 是此项目的主要文件，用于调用各个公用文件来实现具体的功能。具体代码如下。

```
package com.eoeandroid.layout;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class ActivityMain extends Activity {
        OnClickListener listener0 = null;
        OnClickListener listener1 = null;
        OnClickListener listener2 = null;
        OnClickListener listener3 = null;
        Button button0;
        Button button1;
        Button button2;
        Button button3;

        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                listener0 = new OnClickListener() {
                        public void onClick(View v) {
                                Intent intent0 = new Intent(ActivityMain.this, ActivityFrameLayout.class);
                                setTitle("FrameLayout");
                                startActivity(intent0);
                        }
                };
                listener1 = new OnClickListener() {
                        public void onClick(View v) {
                                Intent intent1 = new Intent(ActivityMain.this, ActivityRelativeLayout.class);
                                startActivity(intent1);
                        }
                };
                listener2 = new OnClickListener() {
                        public void onClick(View v) {
                                setTitle("在 ActivityLayout");
                                Intent intent2 = new Intent(ActivityMain.this, ActivityLayout.class);
                                startActivity(intent2);

                        }
```

```
                };
                listener3 = new OnClickListener() {
                        public void onClick(View v) {
                                setTitle("TableLayout");
                                Intent intent3 = new Intent(ActivityMain.this, ActivityTableLayout.class);
                                startActivity(intent3);

                        }
                };
                setContentView(R.layout.main);
                button0 = (Button) findViewById(R.id.button0);
                button0.setOnClickListener(listener0);
                button1 = (Button) findViewById(R.id.button1);
                button1.setOnClickListener(listener1);
                button2 = (Button) findViewById(R.id.button2);
                button2.setOnClickListener(listener2);
                button3 = (Button) findViewById(R.id.button3);
                button3.setOnClickListener(listener3);
        }
}
```

在上述代码中，函数 setContentView(R.layout.main)用于实现 Activity 和 main.xml 的关联；button0、button1、button2、button3 代表了 4 个按钮，在上述代码中对这 4 个按钮实现了引用，并给按钮设置了单击监听器，每一个监听器都跳转到一个新的 Activity。

第 3 步：设计布局信息，此功能由文件 main.xml 实现，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical" android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <Button android:id="@+id/button0"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content" android:text="使用 FrameLayout"/>
        <Button android:id="@+id/button1"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content" android:text="使用 RelativeLayout"/>
        <Button android:id="@+id/button2"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content" android:text="LinearLayout 和 RelativeLayout"/>
        <Button android:id="@+id/button3"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content" android:text="使用 TableLayout"/>
</LinearLayout>
```

上述代码是一个典型的 LinearLayout 布局样式。

第 4 步：设计单击第一个按钮 button0 后的处理动作。在本实例中，单击第一个按钮 button0



后会显示一个地图，此界面是一个 FrameLayout 布局。在文件 activity_frame_layout.xml 中，定义了这幅地图的显示样式，即在 FrameLayout 布局中添加了一个图片显示组件 ImageView 元素，具体实现代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout android:id="@+id/left"
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"                    /**x 轴方向填充空间*/
      android:layout_height="fill_parent"                   /**y 轴方向填充空间*/
>
      <ImageView android:id="@+id/photo"                    /**定义组件的 id*/
            android:src="@drawable/bg"
            android:layout_width="wrap_content"             /**宽度能包容图片*/
            android:layout_height="wrap_content"            /**高度能包容图片*/
  />
</FrameLayout>
```

在上述代码中，可以通过“android:id”来访问定义的这个元素；“android:layout_width="fill_parent"”表示 FrameLayout 布局可以在 x 轴方向填充的空间，“android:layout_height="fill_parent"”表示 FrameLayout 布局可以在 x 轴方向填充的空间；“android:layout_width="wrap_content"”和“android:layout_height="wrap_content"”表示 ImageView 只需将图片完全包含即可。

第 5 步：设计单击第二个按钮 button1 后的处理动作。在本实例中，单击第二个按钮 button1 后会显示要求输入用户名的表单，此功能是通过 RelativeLayout 实现的，对应文件 relative_layout.xml 的具体实现代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<!-- Demonstrates using a relative layout to create a form -->

<RelativeLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent" android:layout_height="wrap_content"
      android:background="@drawable/blue" android:padding="10dip">

      <TextView android:id="@+id/label" android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:text="请输入用户名：" />
      <!--
            这个 EditText 放置在上边 id 为 label 的 TextView 的下边
      -->
      <EditText android:id="@+id/entry" android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:background="@android:drawable/editbox_background"
            android:layout_below="@id/label" />


      <!--
```

```
        取消按钮和容器的右边齐平，并且设置左边的边距为 10dip
    -->
    <Button android:id="@+id/cancel" android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:layout_below="@id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dip" android:text="取消" />

    <!--
        确定按钮在取消按钮的左侧，并且和取消按钮的高度齐平
    -->
    <Button android:id="@+id/ok" android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/cancel"
        android:layout_alignTop="@id/cancel" android:text="确定" />
</RelativeLayout>
```

在上述代码中，主要参数的具体说明如下。

1）android:id：定义组件的 ID。

2）android:layout_width：设置组件的宽度，主要有如下 2 种方式。

- fill_parent：填充父容器。
- wrap_content：仅仅包容住内容即可。

3）android:layout_height：定义组件的高度。

4）android:background="@drawable/blue"：定义组件的背景，在此设置了背景颜色。

5）android:padding="10dip"："dip"表示依赖于设备的像素，有如下 2 种表现方式。

- padding：填充。
- margin：边框。

6）android:layout_below="@id/label"：将此组件放置于 ID 为 label 的组件的下方。这种方式是经典的布局方式，这种方式的好处是不用关心具体的细节，并且适配性很强，在不同屏幕、不同手机设备上都是通用的。

7）android:layout_alignParentRight="true"：也是相对布局，表示和父容器的右边对齐。

8）android:layout_marginLeft="10dip"：设置 ID 为 cancel 的 Button 的左边距为 10dip。

9）android:layout_toLeftOf="@id/cancel"：设置此组件在 ID 为 cancel 的组件的左边。

10）android:layout_alignTop="@id/cancel"：设置此组件和 ID 为 cancel 的组件的高度对齐。

第 6 步：设计单击第三个按钮 button2 后的处理动作。在本实例中，单击第三个按钮 button2 后会显示一系列的文本，此功能是通过 LinearLayout 和 RelativeLayout 联合实现的。具体实现流程如下。

1）第 a 组第 a 项和第 a 组第 b 项：通过 RelativeLayout 实现的，此布局功能是通过文件 left.xml 定义的，具体实现代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
android:id="@+id/left"
    xmlns:android="http://schemas.android.com/apk/res/android"
```



```
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <TextView android:id="@+id/view1" android:background="@drawable/blue"
                android:layout_width="fill_parent"
                android:layout_height="50px" android:text="第 a 组第 a 项" />
        <TextView android:id="@+id/view2"
                android:background="@drawable/yellow"
                android:layout_width="fill_parent"
                android:layout_height="50px" android:layout_below="@id/view1"
                android:text="第 a 组第 b 项" />
</RelativeLayout>
```

在上述代码中，使用了两个 TextView，高度都是 50 像素。此处 TextView 的具体说明如下。

- 第一个 TextView：通过"@drawable/blue"设置其背景颜色是“blue”。
- 第二个 TextView：通过 android:layout_below="@id/view1"设置其位置位与第一个 TextView 的下方。

2）第 b 组第 a 项和第 b 组第 b 项：是通过另外一个 RelativeLayout 实现的，此布局功能是通过文件 right.xml 定义的，具体实现代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout android:id="@+id/right"
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <TextView android:id="@+id/right_view1"
                android:background="@drawable/yellow" android:layout_width="fill_parent"
                android:layout_height="wrap_content" android:text="第 b 组第 a 项" />
        <TextView android:id="@+id/right_view2"
                android:background="@drawable/blue"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_below="@id/right_view1" android:text="第 b 组第 b 项" />
</RelativeLayout>
```

上述代码和文件 left.xml 的类似，在此将不再进行详细介绍。

3）实现 Layout 和 Activity 的关联：即实现一个 Layout 和一个 Activity 的关联，而此 Layout 是在 XML 文件中被定义的。在 Activity 中，为使用方便可以自行构建一个 Layout。根据上述描述编写文件 ActivityLayout.java，主要实现代码如下所示。

```
public class ActivityLayout extends Activity {
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                /**创建一个 Layout **/
                LinearLayout layoutMain = new LinearLayout(this);
```

```
            layoutMain.setOrientation(LinearLayout.HORIZONTAL);
            /**实现 Layout 和 Activity 的关联**/
            setContentView(layoutMain);
            /**得到一个 LayoutInflater 对象，此对象可以对 XML 布局文件进行解析，并生成一个 view**/
            LayoutInflater inflate = (LayoutInflater) getSystemService(Context.LAYOUT_INFLATER_SERVICE);
            RelativeLayout layoutLeft = (RelativeLayout) inflate.inflate(
                    R.layout.left, null);
            RelativeLayout layoutRight = (RelativeLayout) inflate.inflate(
                    R.layout.right, null);
          /**生成一个可以供 Layout 使用的 LayoutParams**/
            RelativeLayout.LayoutParams relParam = new RelativeLayout.LayoutParams(
                    RelativeLayout.LayoutParams.WRAP_CONTENT,
                    RelativeLayout.LayoutParams.WRAP_CONTENT);
    /**将 layoutLeft 添加到 layoutMain，第一个参数是添加进去的 view，第二、三个分别是 view 的高
    度和宽度**/
    layoutMain.addView(layoutLeft, 100, 100);
    /**将 layoutRight 添加到 layoutMain，第二个参数是一个 RelativeLayout.LayoutParams**/
    layoutMain.addView(layoutRight, relParam);
        }
```

第 7 步：设计单击第四个按钮 button3 后的处理动作。在本实例中，单击第四个按钮 button3 后会显示一个整齐排列的表单，此功能是通过 TableLayout 实现的，对应文件 activity_table_layout.xml 的具体实现代码如下。

```
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:stretchColumns="1">
    <TableRow>
        <TextView android:text="用户名:" android:textStyle="bold"
            android:gravity="right" android:padding="3dip" />
        <EditText android:id="@+id/username" android:padding="3dip"
            android:scrollHorizontally="true" />
    </TableRow>
    <TableRow>
        <TextView android:text="密码:" android:textStyle="bold"
            android:gravity="right" android:padding="3dip" />
        <EditText android:id="@+id/password" android:password="true"
            android:padding="3dip" android:scrollHorizontally="true" />
    </TableRow>
    <TableRow android:gravity="right">
        <Button android:id="@+id/cancel"
            android:text="取消" />
        <Button android:id="@+id/login"
            android:text="登录" />
    </TableRow>
</TableLayout>
```



在上述代码中，首先通过标签“TableLayout”定义了一个表格布局，然后通过“TableRow”标签定义了表格布局里的一行，用户可以根据需要在每一行中加入自己需要的一些组件。

6.2.2　效果演示

经过上一小节的介绍，整个实例的主要步骤介绍完毕，在下面的内容中，开始演示本实例的执行效果，具体流程如下。

第 1 步：打开 Eclipse，打开开发完毕的项目文件，右键单击项目名“buju”，在弹出命令中选择“Run As”→“Android Application”命令后开始编译运行当前项目，如图 6-9 所示。



图 6-9　开始编译

第 2 步：运行后的初始效果如图 6-10 所示。



图 6-10　初始效果

第 3 步：单击“使用 FrameLayout”按钮后会显示指定的图片，效果如图 6-11 所示。

第 4 步：单击“演示”按钮后会显示输入用户名界面，效果如图 6-12 所示。



图 6-11　显示图片



图 6-12　显示输入用户名

第 5 步：单击“使用 LinearLayout 和 RelativeLayout”按钮后会显示 4 块不同样式的区域块，效果如图 6-13 所示。

第 6 步：单击“使用 TableLayout”按钮后会显示用户登录表单，效果如图 6-14 所示。



图 6-13　4 块不同样式的区域块



图 6-14　用户登录表单

6.3　Widget组件详解

经过上一节的介绍，了解了 Android 中基本组件的使用方法。在接下来的内容中，将进一步介绍 UI 组件的知识。在本节的内容中，将首先讲解 Widget 包的基本知识。在 Widget 包内包含了按钮、列表框、进度条和图片等常用的控件。

6.3.1　创建一个Widget组件

在下面的内容中，将通过一个实例的实现过程介绍创建一个 widget 组件的过程。本实例保存在“光盘:\daima\6\”中。创建一个 widget 组件的基本流程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“widgetshiyong”的工程文件，如图 6-15 所示。

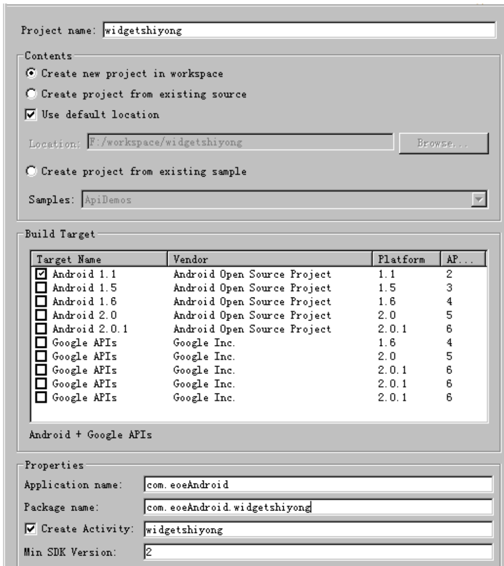


图 6-15　新建一个项目



第 2 步：创建完毕后会自行创建一个 MainActivity，这是应用程序的入口，用户可以打开对应的文件 widgetshiyong.java，其主要代码如下所示。

```
package com.eoeAndroid.widgetshiyong;
import android.app.Activity;
import android.os.Bundle;
public class widgetshiyong extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

在上述代码中，主要功能是 onCreate 方法实现的，上述文件已经关联了一个模板文件 main.xml。这样，就可以在里面继续添加需要的控件了，如按钮、列表框、进度条和图片等。

6.3.2　使用按钮Button

Button 是一个按钮控件，在日常应用时，当单击 Button 后会触发一个事件，这个事件会实现用户需要的功能。例如，用户输入一些信息，单击“确定”或“取消”按钮后，会实现对应的功能。在下面的内容中，将通过一个实例的实现过程介绍 Button 控件的使用过程。本实例的具体实现流程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“widgetbutton”的工程文件。

第 2 步：修改 main.xml 布局，添加一个 TextView 和一个 Button。主要代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:id="@+id/show_TextView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
<Button
    android:id="@+id/Click_Button"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="点击"
    />
```

```
</LinearLayout>
```

第 3 步：在 mainActivity.java 中 findViewByID()获取 TextView 和 Button 资源。主要代码如下所示。

```
show= (TextView)findViewById(R.id.show_TextView);
press=(Button)findViewById(R.id.Click_Button);
```

第 4 步：给 Button 添加事件监听器 Button.OnClickListener()，主要代码如下所示。

```
press.setOnClickListener(new Button.OnClickListener(){
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
            }
        });
```

第 5 步：定义处理事件处理程序，主要代码如下所示。

```
press.setOnClickListener(new Button.OnClickListener(){
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                show.setText("哎呦，button 被点了一下");
            }
        });
```

程序运行后，将首先显示一个“按钮+文本”界面，当单击按钮后会执行单击事件，显示对应的文本提示，如图 6-16 所示。



图 6-16　执行效果

6.3.3　使用文本框TextView

文本框控件 TextView 是使用最频繁的控件之一，在前面的章节中，已经多次使用了 TextView。下面将详细地讲解 TextView 控件的详细使用过程。

1．使用TextView

使用 TextView 需要通过如下 6 个步骤。

第 1 步：导入 TextView 包，代码如下。

```
import android.widget.TextView;
```



第 2 步：在 mainActivity.java 中声明一个 TextView，代码如下。

```
private TextView mTextView01;
```

第 3 步：在 main.xml 中定义一个 TextView，代码如下。

```
<TextView android:text="TextView01"
                android:id="@+id/TextView01"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_x="61px"
                android:layout_y="69px">
</TextView>
```

第 4 步：利用 findViewById()方法获取 main.xml 中的 TextView，代码如下。

```
mTextView01 = (TextView) findViewById(R.id.TextView01);
```

第 5 步：设置 TextView 标签内容，代码如下。

```
String str_2 = "欢迎来到 Android 的 TextView 世界...";
mTextView01.setText(str_2);
```

第 6 步：设置文本超级链接，代码如下。

```
<TextView
        android:id="@+id/TextView02"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:autoLink="all"
        android:text="请访问 Android 开发者：
        http://developer.android.com/index.html">
</TextView>
```

2．使用TextView实现颜色变幻

可以使用 TextView 实现颜色变幻效果，颜色说明如下。

- Color.BLACK：黑色。
- Color.BLUE：蓝色。
- Color.CYAN：青绿色。
- Color.DKGRAY：灰黑色。
- Color.GRAY：灰色。
- Color.GREEN：绿色。
- Color.LTGRAY：浅灰色。
- Color.MAGENTA：红紫色。
- Color.RED：红色。
- Color.TRANSPARENT：透明。

- ❑ Color.WHITE：白色。
- ❑ Color.YELLOW：黄色。

具体实现流程如下。

第 1 步：新建一个工程。

第 2 步：修改 mainActivity.java 文件，添加 12 个 TextView 对象变量，一个 LinearLayout 对象变量、一个 WC 整数变量、一个 LinearLayout.LayoutParams 变量。主要实现代码如下所示。

```
package zyf.ManyColorME;
/*导入要使用的包*/
import android.app.Activity;
import android.graphics.Color;
import android.os.Bundle;
import android.widget.LinearLayout;
import android.widget.TextView;
public class ManyColorME extends Activity {
    /** Called when the activity is first created. */
    /* 定义使用的对象 */
    private LinearLayout myLayout;
    private LinearLayout.LayoutParams layoutP;
    private int WC = LinearLayout.LayoutParams.WRAP_CONTENT;
    private TextView black_TV, blue_TV, cyan_TV, dkgray_TV,
                gray_TV, green_TV,ltgray_TV, magenta_TV, red_TV,
                transparent_TV, white_TV, yellow_TV;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        /* 实例化一个 LinearLayout 布局对象 */
        myLayout = new LinearLayout(this);
        /* 设置 LinearLayout 的布局为垂直布局 */
        myLayout.setOrientation(LinearLayout.VERTICAL);
        /* 设置 LinearLayout 布局背景图片 */
        myLayout.setBackgroundResource(R.drawable.back);
        /* 加载主屏布局 */
        setContentView(myLayout);
        /* 实例化一个 LinearLayout 布局参数，用来添加 View */
        layoutP = new LinearLayout.LayoutParams(WC, WC);
        /* 构造实例化 TextView 对象 */
        constructTextView();
        /* 把 TextView 添加到 LinearLayout 布局中 */
        addTextView();
        /* 设置 TextView 文本颜色 */
        setTextViewColor();
        /* 设置 TextView 文本内容 */
        setTextViewText();
    }
    /* 设置 TextView 文本内容 */
```



```
public void setTextViewText() {
        black_TV.setText("黑色");
        blue_TV.setText("蓝色");
        cyan_TV.setText("青绿色");
        dkgray_TV.setText("灰黑色");
        gray_TV.setText("灰色");
        green_TV.setText("绿色");
        ltgray_TV.setText("浅灰色");
        magenta_TV.setText("红紫色");
        red_TV.setText("红色");
        transparent_TV.setText("透明");
        white_TV.setText("白色");
        yellow_TV.setText("黄色");
}
/*  设置 TextView 文本颜色  */
public void setTextViewColor() {
        black_TV.setTextColor(Color.BLACK);
        blue_TV.setTextColor(Color.BLUE);
        dkgray_TV.setTextColor(Color.DKGRAY);
        gray_TV.setTextColor(Color.GRAY);
        green_TV.setTextColor(Color.GREEN);
        ltgray_TV.setTextColor(Color.LTGRAY);
        magenta_TV.setTextColor(Color.MAGENTA);
        red_TV.setTextColor(Color.RED);
        transparent_TV.setTextColor(Color.TRANSPARENT);
        white_TV.setTextColor(Color.WHITE);
        yellow_TV.setTextColor(Color.YELLOW);
}
/*  构造实例化 TextView 对象  */
public void constructTextView() {
        black_TV = new TextView(this);
        blue_TV = new TextView(this);
        cyan_TV = new TextView(this);
        dkgray_TV = new TextView(this);
        gray_TV = new TextView(this);
        green_TV = new TextView(this);
        ltgray_TV = new TextView(this);
        magenta_TV = new TextView(this);
        red_TV = new TextView(this);
        transparent_TV = new TextView(this);
        white_TV = new TextView(this);
        yellow_TV = new TextView(this);
}
/*  把 TextView 添加到 LinearLayout 布局中  */
public void addTextView() {
        myLayout.addView(black_TV, layoutP);
```

```
            myLayout.addView(blue_TV, layoutP);
            myLayout.addView(cyan_TV, layoutP);
            myLayout.addView(dkgray_TV, layoutP);
            myLayout.addView(gray_TV, layoutP);
            myLayout.addView(green_TV, layoutP);
            myLayout.addView(ltgray_TV, layoutP);
            myLayout.addView(magenta_TV, layoutP);
            myLayout.addView(red_TV, layoutP);
            myLayout.addView(transparent_TV, layoutP);
            myLayout.addView(white_TV, layoutP);
            myLayout.addView(yellow_TV, layoutP);
        }
    }
```

执行后效果如图 6-17 所示。



图 6-17　执行效果

3. 使用TextView实现静态域字体

在计算机系统中，使用 Typeface 来表示字体的风格，具体来说有两种类型。

1）int Style 类型，具体说明如表 6-2 所示。

表 6-2　int Style 类型说明

字　　体	说　　明
BOLD	粗体
BOLD_ITALIC	粗斜体
ITALIC	斜体
NORMAL	普通字体

2）Typeface 类型，具体说明如表 6-3 所示。

表 6-3　Typeface 类型说明

字　　体	说　　明
DEFAULT	默认字体
DEFAULT_BOLD	默认粗体
MONOSPACE	单间隔字体
SANS_SERIF	无衬线字体
SERIF	衬线字体



下面开始讲解编程实现上述字体，首先修改 mainActivity.java，实现多种字体样式的显示。具体实现代码如下所示。

```
package zyf.TypefaceStudy;
/*导入要使用的包*/
import android.app.Activity;
import android.graphics.Color;
import android.graphics.Typeface;
import android.os.Bundle;
import android.view.ViewGroup;
import android.widget.LinearLayout;
import android.widget.TextView;
public class TypefaceStudy extends Activity {
    /** Called when the activity is first created. */
    /*
     * android.graphics.Typeface java.lang.Object
       Typeface 类指定一个字体的字体和固有风格
     * 该类用于绘制，与可选绘制设置一起使用
    如 textSize, textSkewX, textScaleX 当绘制(测量)时来指定如何显示文本
     */
    /* 定义实例化一个布局大小，用来添加 TextView */
    final int WRAP_CONTENT = ViewGroup.LayoutParams.WRAP_CONTENT;
    /* 定义 TextView 对象 */
    private TextView bold_TV, bold_italic_TV, default_TV,
                    default_bold_TV,italic_TV,monospace_TV,
                    normal_TV,sans_serif_TV,serif_TV;
    /* 定义 LinearLayout 布局对象 */
    private LinearLayout linearLayout;
    /* 定义 LinearLayout 布局参数对象 */
    private LinearLayout.LayoutParams linearLayouttParams;
    @Override
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        /* 定义实例化一个 LinearLayout 对象 */
        linearLayout = new LinearLayout(this);
        /* 设置 LinearLayout 布局为垂直布局 */
        linearLayout.setOrientation(LinearLayout.VERTICAL);
        /*设置布局背景图*/
        linearLayout.setBackgroundResource(R.drawable.back);
        /* 加载 LinearLayout 为主屏布局，显示 */
        setContentView(linearLayout);
        /* 定义实例化一个 LinearLayout 布局参数 */
        linearLayouttParams =
          new LinearLayout.LayoutParams(WRAP_CONTENT,WRAP_CONTENT);
        constructTextView();
        setTextSizeOf();
```

```
        setTextViewText() ;
        setStyleOfFont();
        setFontColor();
        toAddTextViewToLayout();
    }
    public void constructTextView() {
        /* 实例化 TextView 对象 */
        bold_TV = new TextView(this);
        bold_italic_TV = new TextView(this);
        default_TV = new TextView(this);
        default_bold_TV = new TextView(this);
        italic_TV = new TextView(this);
        monospace_TV=new TextView(this);
        normal_TV=new TextView(this);
        sans_serif_TV=new TextView(this);
        serif_TV=new TextView(this);
    }
    public void setTextSizeOf() {
        /* 设置绘制的文本大小，该值必须大于 0*/
        bold_TV.setTextSize(24.0f);
        bold_italic_TV.setTextSize(24.0f);
        default_TV.setTextSize(24.0f);
        default_bold_TV.setTextSize(24.0f);
        italic_TV.setTextSize(24.0f);
        monospace_TV.setTextSize(24.0f);
        normal_TV.setTextSize(24.0f);
        sans_serif_TV.setTextSize(24.0f);
        serif_TV.setTextSize(24.0f);
    }
    public void setTextViewText() {
        /* 设置文本 */
        bold_TV.setText("BOLD");
        bold_italic_TV.setText("BOLD_ITALIC");
        default_TV.setText("DEFAULT");
        default_bold_TV.setText("DEFAULT_BOLD");
        italic_TV.setText("ITALIC");
        monospace_TV.setText("MONOSPACE");
        normal_TV.setText("NORMAL");
        sans_serif_TV.setText("SANS_SERIF");
        serif_TV.setText("SERIF");
    }
    public void setStyleOfFont() {
        /* 设置字体风格 */
        bold_TV.setTypeface(null, Typeface.BOLD);
        bold_italic_TV.setTypeface(null, Typeface.BOLD_ITALIC);
        default_TV.setTypeface(Typeface.DEFAULT);
```



```
                default_bold_TV.setTypeface(Typeface.DEFAULT_BOLD);
                italic_TV.setTypeface(null, Typeface.ITALIC);
                monospace_TV.setTypeface(Typeface.MONOSPACE);
                normal_TV.setTypeface(null, Typeface.NORMAL);
                sans_serif_TV.setTypeface(Typeface.SANS_SERIF);
                serif_TV.setTypeface(Typeface.SERIF);
        }
        public void setFontColor() {
                /* 设置文本颜色 */
                bold_TV.setTextColor(Color.BLACK);
                bold_italic_TV.setTextColor(Color.CYAN);
                default_TV.setTextColor(Color.GREEN);
                default_bold_TV.setTextColor(Color.MAGENTA);
                italic_TV.setTextColor(Color.RED);
                monospace_TV.setTextColor(Color.WHITE);
                normal_TV.setTextColor(Color.YELLOW);
                sans_serif_TV.setTextColor(Color.GRAY);
                serif_TV.setTextColor(Color.LTGRAY);
        }
        public void toAddTextViewToLayout() {
                /* 把 TextView 加入 LinearLayout 布局中 */
                linearLayout.addView(bold_TV, linearLayouttParams);
                linearLayout.addView(bold_italic_TV, linearLayouttParams);
                linearLayout.addView(default_TV, linearLayouttParams);
                linearLayout.addView(default_bold_TV, linearLayouttParams);
                linearLayout.addView(italic_TV, linearLayouttParams);
                linearLayout.addView(monospace_TV, linearLayouttParams);
                linearLayout.addView(normal_TV, linearLayouttParams);
                linearLayout.addView(sans_serif_TV, linearLayouttParams);
                linearLayout.addView(serif_TV, linearLayouttParams);
        }
}
```

运行后效果如图 6-18 所示。

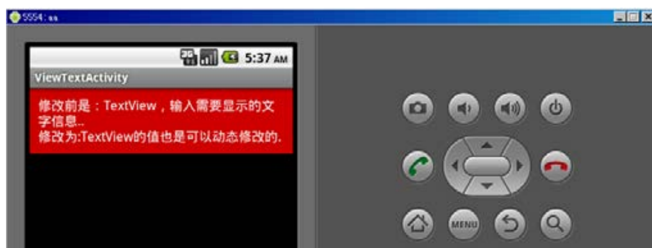


图 6-18　运行效果

4．在代码中更改TextView 文字颜色

在开发过程中，可以在代码中更改 TextView 文字颜色，实现流程如下。

第 1 步：在文件 main.xml 中编写布局代码，代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
<TextView
    android:text="TextView01"
    android:id="@+id/TextView01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
</TextView>
<TextView
    android:text="这里使用 Graphics 颜色静态常量"
    android:id="@+id/TextView02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
</TextView>
</LinearLayout>
```

第 2 步：新加文件 drawable.xml，在此添加一个 white 颜色值，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <color name="white">#ffffffff</color>
</resources>
```

第 3 步：在代码中由 ID 获取 TextView，具体代码如下所示。

```
TextView text_A=(TextView)findViewById(R.id.TextView01);
TextView text_B=(TextView)findViewById(R.id.TextView02);
```

第 4 步：获取 Resources 对象，具体代码如下所示。

```
Resources myColor_R=getBaseContext().getResources();
//getBaseContext()获得基础 Context
//getResources()从 Context 获取资源实例对象
```

第 5 步：获取 Drawable 对象，具体代码如下所示。

```
Drawable myColor_D=myColor_R.getDrawable(R.color.white);
```



第 6 步：设置文本背景颜色，具体代码如下所示。

```
text_A.setBackgroundDrawable(myColor_D);
```

第 7 步：利用 android.graphics.Color 的颜色静态变量来改变文本颜色，具体代码如下所示。

```
text_A.setTextColor(android.graphics.Color.GREEN);
```

第 8 步：利用 Color 的静态常量来设置文本颜色，具体代码如下所示。

```
text_B.setTextColor(Color.RED);
```

6.3.4　使用编辑框EditText

编辑框控件 EditText 的用法和 TextView 类似，它能生成一个可编辑的文本框。使用编辑框控件 EditText 的基本流程如下。

第 1 步：在程序的主窗口界面中添加一个 EditText 按钮，然后设定其监听器在接收到单击事件时，程序打开 EditText 的界面。文件 editview.xml 的具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
//供用户输入值
<EditText android:id="@+id/edit_text"
android:layout_width="fill_parent"
android:layout_height="wrap_content"
android:text="这里可以输入文字" />
   //用于获取输入的值
     <Button android:id="@+id/get_edit_view_button"
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="获取 EditView 的值" />
</LinearLayout>
```

第 2 步：编写事件处理文件 EditTextActivity.java，主要代码如下所示。

```
package com.eoemobile.book.ex_widgetdemo;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.CheckBox;
import android.widget.EditText;
```

```
import android.widget.TextView;
public class EditTextActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTitle("EditTextActivity");
        setContentView(R.layout.editview);
        find_and_modify_text_view();
    }
    private void find_and_modify_text_view() {
        Button get_edit_view_button = (Button) findViewById(R.id.get_edit_view_button);
        get_edit_view_button.setOnClickListener(get_edit_view_button_listener);
    }

    private Button.OnClickListener get_edit_view_button_listener = new Button.OnClickListener() {
      /**响应代码，显示 EditText 中的值**/
        public void onClick(View v) {
            EditText edit_text = (EditText) findViewById(R.id.edit_text);
            CharSequence edit_text_value = edit_text.getText();
            setTitle("EditText 的值:"+edit_text_value);
        }
    };
}
```

执行后，将首先显示默认的文本和输入框，如图 6-19 所示；输入一段文本，单击“获取 EditView 的值”按钮后，会获取输入的文字，并显示出输入的文字，如图 6-20 所示。

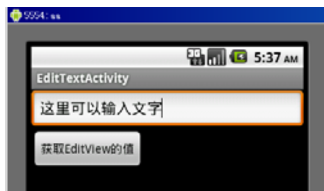


图 6-19　初始效果

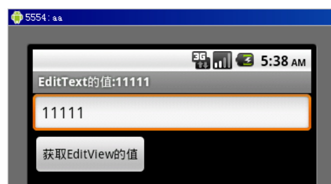


图 6-20　运行效果

6.3.5　使用多项选择控件CheckBox

CheckBox 控件能够为用户提供输入信息，用户可以一次性选择多个选项。在 Android 中，使用 CheckBox 控件也需要在 XML 文件中定义，具体使用流程如下。

第 1 步：设计 XML 文件 check_box.xml，在里面插入 4 个选项供用户选择，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
```



```
        android:layout_height="fill_parent"
    >

    <CheckBox android:id="@+id/plain_cb"
        android:text="Plain"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
    />

    <CheckBox android:id="@+id/serif_cb"
        android:text="Serif"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:typeface="serif"
    />

    <CheckBox android:id="@+id/bold_cb"
        android:text="Bold"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textStyle="bold"
    />

    <CheckBox android:id ="@+id/italic_cb"
        android:text="Italic"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textStyle="italic"
    />

        <Button android:id="@+id/get_view_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="获取 CheckBox 的值" />

    </LinearLayout>
```

在上述代码中分别创建了 4 个 CheckBox 选项供用户选择，然后插入了一个 Button 控件，供用户选择单击后处理特定事件。

第 2 步：编写事件处理文件 CheckBoxActivity.java 的代码，主要代码如下所示。

```
    @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setTitle("CheckBoxActivity");
            setContentView(R.layout.check_box);
```

```
            find_and_modify_text_view();
        }

        private void find_and_modify_text_view() {
            plain_cb = (CheckBox) findViewById(R.id.plain_cb);
            serif_cb = (CheckBox) findViewById(R.id.serif_cb);
            italic_cb = (CheckBox) findViewById(R.id.italic_cb);
            bold_cb = (CheckBox) findViewById(R.id.bold_cb);
            Button get_view_button = (Button) findViewById(R.id.get_view_button);
            get_view_button.setOnClickListener(get_view_button_listener);
        }

        private Button.OnClickListener get_view_button_listener = new Button.OnClickListener() {
            public void onClick(View v) {
                String r = "";
                if (plain_cb.isChecked()) {
                    r = r + "," + plain_cb.getText();
                }
                if (serif_cb.isChecked()) {
                    r = r + "," + serif_cb.getText();
                }
                if (italic_cb.isChecked()) {
                    r = r + "," + italic_cb.getText();
                }
                if (bold_cb.isChecked()) {
                    r = r + "," + bold_cb.getText();
                }
                setTitle("Checked: " + r);
            }
        };
    }
```

在上述代码中，把用户选中的选项值显示在 Title 上面。

执行后，将首先显示 4 个选项值供用户选择，如图 6-21 所示；用户选择某些选项并单击“获取 CheckBox 的值”按钮后，文本提示用户选择的选项，如图 6-22 所示。

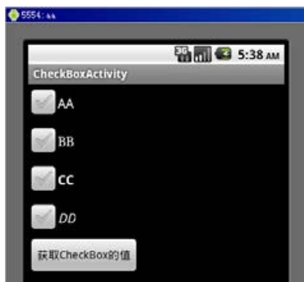


图 6-21　初始效果

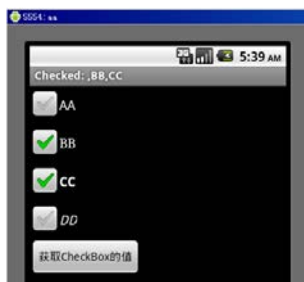


图 6-22　运行效果



6.3.6 使用单项选择控件RadioGroup

单项选择控件 RadioGroup 是和多项选择控件 CheckBox 相对应的，但是它只能供用户选择一个选项。在 Android 中，使用 CheckBox 控件也需要在 XML 文件中定义，具体使用流程如下。

第 1 步：设计 XML 文件 radio_group.xml，在里面插入 4 个选项供用户选择，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:orientation="vertical">
    <RadioGroup
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:orientation="vertical"
        android:checkedButton="@+id/lunch"
        android:id="@+id/menu">
        <RadioButton
            android:text="AA"
            android:id="@+id/breakfast"
            />
        <RadioButton
            android:text="BB"
            android:id="@id/lunch" />
        <RadioButton
            android:text="CC"
            android:id="@+id/dinner" />
        <RadioButton
            android:text="DD"
            android:id="@+id/all" />
    </RadioGroup>
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="清除"
        android:id="@+id/clear" />
</LinearLayout>
```

在上述代码中插入了 1 个 RadioGroup 空间，它提出了 4 个选项供用户选择，然后插入了一个 Button 控件，用于清除掉用户选择的选项。

第 2 步：编写事件处理文件 RadioGroupActivity.java 的代码，主要代码如下所示。

```
@Override
    protected void onCreate(Bundle savedInstanceState) {
```

```
            super.onCreate(savedInstanceState);
            setContentView(R.layout.radio_group);
            setTitle("RadioGroupActivity");
            mRadioGroup = (RadioGroup) findViewById(R.id.menu);
            Button clearButton = (Button) findViewById(R.id.clear);
            clearButton.setOnClickListener(this);
        }
```

当用户单击“清除”按钮后将使用 setTitle 修改 Title 为“RadioGroupActivity”，然后会获取 RadioGroup 对象和按钮对象。

至此，整个实例设计完毕。执行后，将首先显示 4 个选项值供用户选择，如图 6-23 所示；用户选择一个选项并单击“清除”按钮后，将会清除选择的选项，如图 6-24 所示。

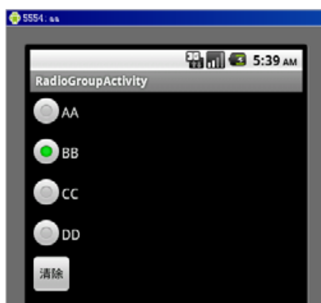


图 6-23　初始效果

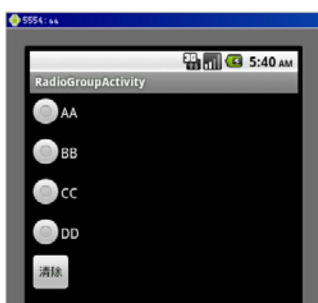


图 6-24　运行效果

6.3.7　使用下拉列表控件Spinner

下拉列表控件 Spinner 能够提供下拉选择样式的输入框，用户不需要输入的数据，只需选择一个选项后即可在框中完成数据输入。使用下拉列表控件 Spinner 控件的具体实现流程如下。

第 1 步：在 Android 中使用 Spinner 控件之前，首先需要在 main.xml 中添加一个按钮，单击这个按钮后会启动这个 SpinnerActivity 文件。

```
<Button android:id="@+id/spinner_button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="Spinner"
/>
```

第 2 步：在文件 MainActivity.java 编写上述按钮的处理事件代码，具体如下。

```
private Button.OnClickListener spinner_button_listener = new Button.OnClickListener() {
        public void onClick(View v) {
            Intent intent = new Intent();
            intent.setClass(MainActivity.this, SpinnerActivity.class);
            startActivity(intent);
        }
    };
```



在上述代码中，启动了 SpinnerActivity，此 SpinnerActivity 可以展示 Spinner 组件的界面。在具体实现上，先创建了 SpinnerActivity 的 Activity，然后修改了其 onCreate 方法，设置其对应模板为 spinner.xml。文件 SpinnerActivity.java 中对应的代码如下。

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTitle("SpinnerActivity");
        setContentView(R.layout.spinner);
        find_and_modify_view();
    }
```

第 3 步：编写文件 spinner.xml，主要代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
    <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Spinner_1"
    />
<Spinner    android:id="@+id/spinner_1"
       android:layout_width="fill_parent"
       android:layout_height="wrap_content"
       android:drawSelectorOnTop="false"
/>
</LinearLayout>
```

在上述代码中，添加了两个 TextView 控件和两个 Spinner 控件。

第 4 步：在文件 AndroidManifest.xml 中添加如下代码。

```
<activity android:name="SpinnerActivity"></activity>
```

在上述代码中，定义的 Spinner 组件的 ID 为 spinner_1，宽度占满了其父元素“LinearLayout”的宽，高度自适应。

经过上述处理后，即可在界面中生成一个简单的单选选项界面，但是在列表中并没有选项值。如果要在下拉列表中实现可供用户选择的选项值，需要在里面填充一些数据。

第 5 步：载入列表数据，首先定义需要载入的数据，然后在 onCreate 方法中通过调用 find_and_modify_view()来完成数据载入。文件 SpinnerActivity.java 中实现上述功能的具体代码如下所示。

```
private static final String[] mCountries = { "China" ,"Russia", "Germany",
                "Ukraine", "Belarus", "USA" };
```

```
        private void find_and_modify_view() {
                spinner_c = (Spinner) findViewById(R.id.spinner_1);
                allcountries = new ArrayList<String>();
                for (int i = 0; i < mCountries.length; i++) {
                    allcountries.add(mCountries[i]);
                }
                aspnCountries = new ArrayAdapter<String>(this,
                        android.R.layout.simple_spinner_item, allcountries);
                aspnCountries
        .setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
                spinner_c.setAdapter(aspnCountries);
```

在上述代码中，将定义的 mCountries 数据载入到了 Spinner 组件中。

第 6 步：在文件 spinner.xml 中预定义数据，即在 spinner.xml 模板中再添加一个 Spinner 组件，具体代码如下所示。

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="Spinner_2 From arrays xml file"
    />
 <Spinner    android:id="@+id/spinner_2"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:drawSelectorOnTop="false"
/>
```

第 7 步：在文件 SpinnerActivity.java 中初始化它的值，具体代码如下所示。

```
spinner_2 = (Spinner) findViewById(R.id.spinner_2);
    ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
    this, R.array.countries, android.R.layout.simple_spinner_item);
    adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
    spinner_2.setAdapter(adapter);
```

在上述代码中，将 R.array.countries 对应值载入到了 spinner_2 中去，而 R.array.countries 的对应值是在文件 array.xml 中预先定义的，文件 array.xml 的具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- Used in Spinner/spinner_2.java -->
    <string-array name="countries">
        <item>China2</item>
        <item>Russia2</item>
        <item>Germany2</item>
        <item>Ukraine2</item>
        <item>Belarus2</item>
```



```
            <item>USA2</item>
        </string-array>
    </resources>
```

在上述代码中，预定义了一个名为“countries”的数组。

至此，整个实例设计完毕。执行后，将首先显示 2 个下拉列表表单，如图 6-25 所示；用户单击一个下拉列表单后面的▼时，会弹出一个由 Spinner 组件实现的下拉选项框，如图 6-26 所示；当选择一个选项后，选项值会自动出现在输入表单中，如图 6-27 所示。



图 6-25　初始效果

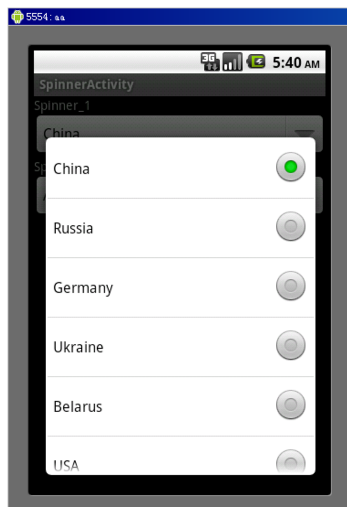


图 6-26　运行效果

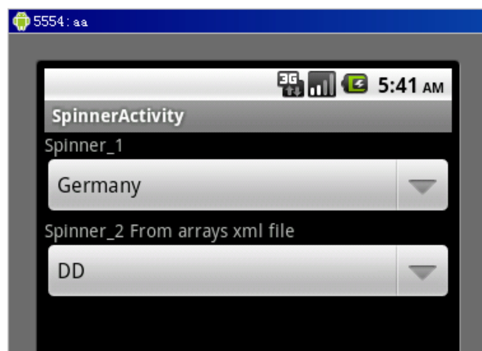


图 6-27　选择值自动出现在表单中

6.3.8　使用自动完成文本控件AutoCompleteTextView

自动完成文本控件 AutoCompleteTextView 的功能是，帮助用户自动输入数据。例如，当用户输入一个字符后，能够根据这个字符提示显示出与之相关的数据。此应用在搜索引擎中比较常见，例如，用户在百度中输入关键字“三枪”后，会在下拉列表中自动显示出相关的关键词，如图 6-28 所示。





图 6-28　百度的输入提示框

在 Android 手机应用中，通过 AutoCompleteTextView 可以实现和图 6-28 类似的功能。使用 AutoCompleteTextView 控件的基本流程如下所示。

第 1 步：修改 main.xml 布局，添加一个 TextView、一个 AutoCompleteTextView、一个 Button，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
    android:id="@+id/widget0"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
>
<TextView
    android:id="@+id/TextView_InputShow"
    android:layout_width="228px"
    android:layout_height="47px"
    android:text="请输入"
    android:textSize="25px"
    android:layout_x="42px"
    android:layout_y="37px"
>
</TextView>
<AutoCompleteTextView
    android:id="@+id/AutoCompleteTextView_input"
    android:layout_width="275px"
    android:layout_height="wrap_content"
    android:text=""
    android:textSize="18sp"
    android:layout_x="23px"
    android:layout_y="98px"
>
</AutoCompleteTextView>
<Button
```



```
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_x="127dip"
        android:text="清空"
        android:id="@+id/Button_clean"
        android:layout_y="150dip">
</Button>
</AbsoluteLayout>
```

第 2 步：修改 mainActivity.java，添加自动完成功能处理事件，具体代码如下所示。

```
package zyf.Ex_Ctrl_13ME;
/*导入使用的包*/
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.AutoCompleteTextView;
import android.widget.Button;
import android.widget.TextView;
public class Ex_Ctrl_13ME extends Activity {
    /** Called when the activity is first created. */
    /*定义要使用的类对象*/
    private String[] normalString =
            new String[] {
            "Android", "Android Blog","Android Market", "Android SDK",
            "Android AVD","BlackBerry","BlackBerry JDE", "Symbian",
            "Symbian Carbide", "Java 2ME","Java FX", "Java 2EE",
            "Java 2SE", "Mobile", "Motorola", "Nokia", "Sun",
            "Nokia Symbian", "Nokia forum", "WindowsMobile", "Broncho",
            "Windows XP", "Google", "Google Android ", "Google 浏览器",
            "IBM", "MicroSoft", "Java", "C++", "C", "C#", "J#", "VB" };
    @SuppressWarnings("unused")
    private TextView show;
    private AutoCompleteTextView autoTextView;
    private Button clean;
    private ArrayAdapter<String> arrayAdapter;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        /*装入主屏布局 main.xml*/
        setContentView(R.layout.main);
        /*从 XML 中获取 UI 元素对象*/
        show = (TextView) findViewById(R.id.TextView_InputShow);
        autoTextView =
          (AutoCompleteTextView) findViewById(R.id.AutoCompleteTextView_input);
        clean = (Button) findViewById(R.id.Button_clean);
```

```
                    /*实现一个适配器对象，用来给自动完成输入框添加自动装入的内容*/
                    arrayAdapter = new ArrayAdapter<String>(this,
                              android.R.layout.simple_dropdown_item_1line, normalString);
                    /*给自动完成输入框添加内容适配器*/
                    autoTextView.setAdapter(arrayAdapter);
                    /*给清空按钮添加点击事件处理监听器*/
                    clean.setOnClickListener(new Button.OnClickListener() {
                         @Override
                         public void onClick(View v) {
                              // TODO Auto-generated method stub
                              /*清空*/
                              autoTextView.setText("");
                         }
                    });
          }
     }
```

经过上述操作，我们成功地使用了 AutoCompleteTextView 控件。执行后，当在表单中输入数据后，会根据预先准备的数据进行提示，如图 6-29 所示。

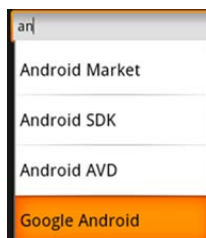


图 6-29　百度的输入提示框

6.3.9　使用日期选择器控件DatePicker

日期选择器控件 DatePicker 的功能是为用户提供快速选择日期的方法。众所周知，日期的格式是“年—月—日”，在很多系统中都为用户提供了日期选择表单，这样不用用户输入具体的日期，只需利用鼠标点击即可完成日期的设置。使用日期选择器控件 DatePicker 的具体流程如下。

第 1 步：在 main.xml 中添加一个按钮，用于打开 DatePicker 界面，具体代码如下。

```
     <Button android:id="@+id/date_picker_button"
     android:layout_width="wrap_content"
     android:layout_height="wrap_content"
     android:text="DatePicker"
     />
```

在上述代码中，定义了一个 ID 为“DatePicker_button”的按钮。

第 2 步：定义上述按钮响应处理事件，具体代码如下所示。



```
private Button.OnClickListener date_picker_button_listener = new Button.OnClickListener() {
        public void onClick(View v) {
            Intent intent = new Intent();
            intent.setClass(MainActivity.this, DatePickerActivity.class);
            startActivity(intent);
        }
    };
```

这样，当单击“DatePicker”按钮后，会跳转到 DatePickerActivity 上。当创建一个 Activity 组件后，需要在其 onCreate 方法中指定需要绑定的模板文件为 date_picker.xml。

第 3 步：在文件 DatePickerActivity.java 中编写 onCreate 的实现代码，具体代码如下所示。

```
public class DatePickerActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTitle("CheckBoxActivity");
        setContentView(R.layout.date_picker);
        DatePicker dp =    (DatePicker)this.findViewById(R.id.date_picker);
        dp.init(2010, 5, 17, null);
    }
```

在上述代码中，设置了开始时间为 2010 年 5 月 17 日。

第 4 步：在文件 date_picker.xml 中添加 DatePicker 组件，主要代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="wrap_content">

    <DatePicker
        android:id="@+id/date_picker"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

在上述代码中，设置了 DatePicker 控件的 ID 为“date_picker”，其宽度和高度都为自适应。

第 5 步：在文件 AndroidManifest.xml 中添加 Activity 申明，具体代码如下。

```
<activity android:name="DatePickerActivity" />
```

至此，整个实例设计完毕。执行后，将首先显示设置的起始日期，如图 6-30 所示；分别单击月、日、年上面的“+”或下面的“-”后，将会自动显示更改后的月、日、年，如图 6-31 所示。

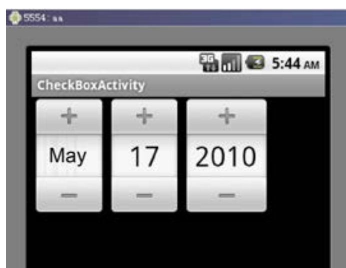


图 6-30　初始效果



图 6-31　改变后效果

6.3.10　使用时间选择器TimePicker控件

时间选择器 TimePicker 控件和 DatePicker 控件的功能类似，其功能是为用户提供快速选择时间的方法。使用时间选择器 TimePicker 控件的基本流程如下。

第 1 步：在 main.xml 文件中添加一个 button 按钮，具体代码如下。

```
<Button android:id="@+id/time_picker_button"
android:layout_width="wrap_content"
android:layout_height="wrap_content"
android:text="TimePicker"
/>
```

第 2 步：编写响应上述按钮“time_picker”的事件代码，具体代码如下。

```
private Button.OnClickListener time_picker_button_listener = new Button.OnClickListener() {
        public void onClick(View v) {
            Intent intent = new Intent();
            intent.setClass(MainActivity.this, TimePickerTimePicker.class);
            startActivity(intent);
        }
};
```

通过上述代码，当单击按钮“time_picker”后会跳转到 TimePickerActivity 上。

第 3 步：创建一个 Activity，然后在其 onCreate 方法中指定需要绑定的模板为 time_picker.xml。文件 TimePickerActivity.java 中 onCreate 方法的实现代码如下。

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTitle("TimePickerActivity");
        setContentView(R.layout.time_picker);
        TimePicker tp =  (TimePicker)this.findViewById(R.id.time_picker);
        tp.setIs24HourView(true);
    }
```

在上述代码中，首先指定了对应的布局模板是 time_picker.xml，然后获取了其中的 TimePicker 控件。

第 4 步：在文件 time_picker.xml 中定义 TimePicker，具体代码如下。



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical" android:layout_width="fill_parent"
        android:layout_height="wrap_content">
<TimePicker
  android:id="@+id/time_picker"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"/>
</LinearLayout>
```

至此，整个实例设计完毕。执行后，将首先显示设置的起始时间，分别单击时间上面的“+”或下面的“–”后，将会自动显示更改后时间，如图 6–32 所示。

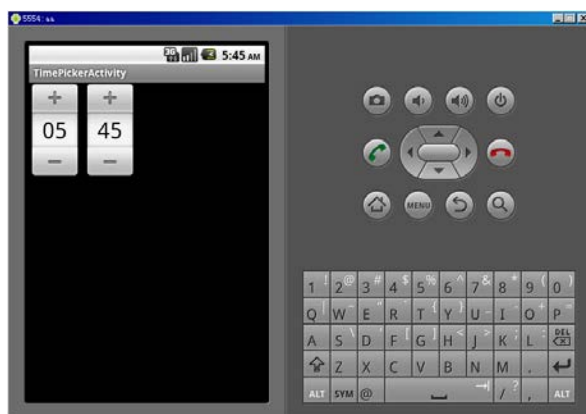


图 6–32　运行效果

6.3.11　联合应用DatePicker和TimePicker

在日常项目应用中，通常会将 DatePicker 和 TimePicker 一块使用。在下面的内容中，将简单讲解联合使用 DatePicker 和 TimePicker 的基本流程，具体流程如下。

第 1 步：修改 main.xml 布局，分别添加一个 DatePicker、一个 TimePicker、一个 TextView。对应代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
      android:id="@+id/widget0"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      xmlns:android="http://schemas.android.com/apk/res/android">
<DatePicker
      android:id="@+id/my_DatePicker"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_x="10px"
      android:layout_y="10px">
```




```
</DatePicker><!-- 日期设置器 -->
<TimePicker
    android:id="@+id/my_TimePicker"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="10px"
    android:layout_y="150px">
</TimePicker><!-- 事件设置器 -->
<TextView
    android:id="@+id/my_TextView"
    android:layout_width="228px"
    android:layout_height="29px"
    android:text="TextView"
    android:layout_x="10px"
    android:layout_y="300px">
</TextView>
</AbsoluteLayout>
```

第 2 步：实现 DatePicker 的初始化与日期改变事件的处理，具体代码如下。

```
/* 定义程序用到的 UI 元素对象:日历设置器*/
        DatePicker my_datePicker;
        /* findViewById()从 XML 中获取 UI 元素对象 */
         my_datePicker = (DatePicker) findViewById(R.id.my_DatePicker);
/*为日历设置器添加点击事件监听器，处理设置日期事件*/
         my_datePicker.init(my_Year, my_Month, my_Day,
                          new DatePicker.OnDateChangedListener(){
             @Override
             public void onDateChanged(DatePicker view, int year,
                      int monthOfYear, int dayOfMonth) {
                 // TODO Auto-generated method stub
                 /*日期改变事件处理*/
             }
         });
```

第 3 步：实现 TimePicker 的初始化与时间改变事件的处理，对应代码如下。

```
/* 定义程序用到的 UI 元素对象:时间设置器*/
        TimePicker my_timePicker;
       /* findViewById()从 XML 中获取 UI 元素对象 */
         my_timePicker = (TimePicker) findViewById(R.id.my_TimePicker);
       /* 把时间设置成 24 小时制 */
         my_timePicker.setIs24HourView(true);
       /*为时间设置器添加点击事件监听器，处理设置时间事件*/
         my_timePicker.setOnTimeChangedListener(new
                                      TimePicker.OnTimeChangedListener(){
             @Override
```



```
            public void onTimeChanged(TimePicker view, int hourOfDay,
                                                                    int minute) {
                // TODO Auto-generated method stub
                /*时间改变事件处理*/
            }
        });
```

第 4 步：修改 mainActivity.java，添加动态修改时间并显示效果，对应代码如下。

```
package zyf.Ex_Ctrl_15ME;
/*导入要使用的包*/
import java.util.Calendar;
import java.util.Locale;
import android.app.Activity;
import android.os.Bundle;
import android.widget.DatePicker;
import android.widget.TextView;
import android.widget.TimePicker;
public class Ex_Ctrl_15ME extends Activity {
    /** Called when the activity is first created. */
    /* 定义时间变量：年、月、日、小时、分钟 */
    int my_Year;
    int my_Month;
    int my_Day;
    int my_Hour;
    int my_Minute;
    /* 定义程序用到的 UI 元素对象:日历设置器、时间设置器、显示时间的 TextView */
    DatePicker my_datePicker;
    TimePicker my_timePicker;
    TextView showDate_Time;
    /* 定义日历对象，初始化时，用来获取当前时间 */
    Calendar my_Calendar;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        /* 从 Calendar 抽象基类获得实例对象，并设置成中国时区 */
        my_Calendar = Calendar.getInstance(Locale.CHINA);
        /* 从日历对象中获取当前的：年、月、日、时、分 */
        my_Year = my_Calendar.get(Calendar.YEAR);
        my_Month = my_Calendar.get(Calendar.MONTH);
        my_Day = my_Calendar.get(Calendar.DAY_OF_MONTH);
        my_Hour = my_Calendar.get(Calendar.HOUR_OF_DAY);
        my_Minute = my_Calendar.get(Calendar.MINUTE);
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        /* findViewById()从 XML 中获取 UI 元素对象 */
        my_datePicker = (DatePicker) findViewById(R.id.my_DatePicker);
        my_timePicker = (TimePicker) findViewById(R.id.my_TimePicker);
```




```
        showDate_Time = (TextView) findViewById(R.id.my_TextView);
        /*  把时间设置成 24 小时制  */
        my_timePicker.setIs24HourView(true);
        /*  显示时间  */
        loadDate_Time();
        /*为日历设置器添加点击事件监听器，处理设置日期事件*/
        my_datePicker.init(my_Year, my_Month, my_Day,
                                        new DatePicker.OnDateChangedListener(){
            @Override
            public void onDateChanged(DatePicker view, int year,
                                        int month Of Year, int day Of  Month) {
                // TODO Auto-generated method stub
                /*把设置改动后的日期赋值给我的日期对象*/
                my_Year=year;
                my_Month=monthOfYear;
                my_Day=dayOfMonth;
                /*  动态显示修改后的日期  */
                loadDate_Time();
            }
        });
        /*为时间设置器添加点击事件监听器，处理设置时间事件*/
        my_timePicker.setOnTimeChangedListener(new
                                        TimePicker.OnTimeChangedListener(){
            @Override
            public void onTimeChanged(TimePicker view, int hourOfDay,
                                                        int minute) {
                /*把设置改动后的时间赋值给我的时间对象*/
                my_Hour=hour Of  Day;
                my_Minute=minute;
                /*  动态显示修改后的时间  */
                loadDate_Time();
            }
        });
    }
    /*  设置显示日期时间的方法  */
    private void loadDate_Time() {
        showDate_Time.setText(new StringBuffer()
               .append(my_Year).append("/")
                .append(FormatString(my_Month + 1))
                .append("/").append(FormatString(my_Day))
                .append("    ").append(FormatString(my_Hour))
                .append(" : ").append(FormatString(my_Minute)));
    }
    /*  日期时间显示两位数的方法  */
    private String FormatString(int x) {
        String s = Integer.toString(x);
```



```
                if (s.length() == 1) {
                    s = "0" + s;
                }
                return s;
            }
        }
```

至此，联合应用 DatePicker 和 TimePicker 的实例设计完毕。执行效果如图 6-33 所示。



图 6-33　运行效果

6.3.12　使用滚动视图控件ScrollView

滚动视图控件 ScrollView 的功能是，能够在手机屏幕中生成一个滚动样式的显示方式，这样即使内容超出了屏幕大小，也能通过滚动的方式供用户浏览。使用滚动视图控件 ScrollView 的方法比较简单，只需在 LinearLayout 外面增加一个 ScrollView 即可，代码如下。

```
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
>
```

在上述代码中，将滚动视图控件 ScrollView 放在了 LinearLayout 的外面，这样当 LinearLayout 中的内容超过屏幕大小时，会实现滚动浏览功能。程序运行后的效果如图 6-34 所示。

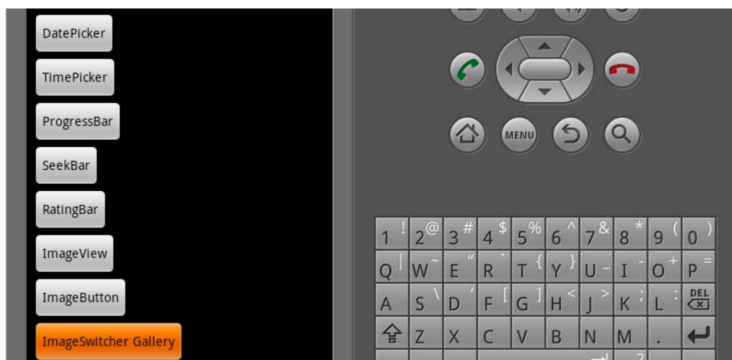


图 6-34　运行效果

6.3.13　使用进度条控件ProgressBar

进度条控件 ProgressBar 的功能是，以图像化的方式显示某个过程的进度，这样做的好处是能够更加直观地显示进度。进度条在计算机领域中非常常见，例如，软件安装过程一般使用进度条。使用进度条控件 ProgressBar 的基本流程如下。

第 1 步：在 main.xml 中增加一个按钮，对应代码如下所示。

```
<Button android:id="@+id/progress_bar_button"
```

```
				android:layout_width="wrap_content"
				android:layout_height="wrap_content"
				android:text="ProgressBar"
		/>
```

第 2 步：编写单击按钮的事件处理程序，用于打开进度条界面。文件 MainActivity.java 中的对应代码如下所示。

```
		private Button.OnClickListener progress_bar_button_listener = new Button.OnClickListener() {
				public void onClick(View v) {
					Intent intent = new Intent();
					intent.setClass(MainActivity.this, ProgressBarActivity.class);
					startActivity(intent);
				}
			};
```

通过上述代码，当单击按钮后会启动 ProgressBarActivity。

第 3 步：编写文件 ProgressBarActivity.java，用于设置其对应的布局文件为 Progress_Bar.xml，具体代码如下所示。

```
		public class ProgressBarActivity extends Activity {
			CheckBox plain_cb;
			CheckBox serif_cb;
			CheckBox italic_cb;
			CheckBox bold_cb;
			@Override
			public void onCreate(Bundle savedInstanceState) {
				super.onCreate(savedInstanceState);
				setTitle("ProgressBarActivity");
				setContentView(R.layout.progress_bar);
			}
```

第 4 步：编写文件 Progress_Bar.xml，在里面插入 2 个 ProgressBar 控件，对应代码如下所示。

```
		<?xml version="1.0" encoding="utf-8"?>
		<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
			android:orientation="vertical" android:layout_width="fill_parent"
			android:layout_height="wrap_content">
		<TextView
				android:layout_width="wrap_content"
				android:layout_height="wrap_content"
				android:text="圆形进度条" />
		<ProgressBar
		    android:id="@+id/progress_bar"
		  android:layout_width="wrap_content"
		  android:layout_height="wrap_content"/>
```



```
<TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="水平进度条" />
<ProgressBar android:id="@+id/progress_horizontal"
        style="?android:attr/progressBarStyleHorizontal"
        android:layout_width="200dip"
        android:layout_height="wrap_content"
        android:max="100"
        android:progress="50"
        android:secondaryProgress="75" />
</LinearLayout>
```

通过上述代码，插入了 2 个 ProgressBar 控件。其中，设置第一个是环形进度条，设置第二个是水平进度样式。然后设置第一个进度到 50，第二个进度到 75。

至此，整个实例设计完毕。执行后将显示对应样式的进度条，如图 6-35 所示。

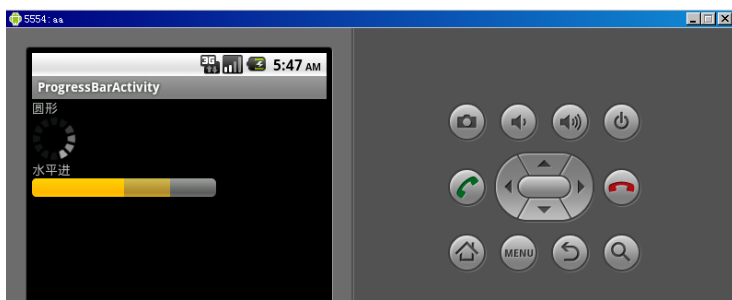


图 6-35　运行效果

6.3.14　使用拖动条控件SeekBar

拖动条控件 SeekBar 的功能是，通过拖动某个进程来直观地显示进度。现实中最常见的拖动条应用是播放器的播放进度，用户可以通过拖动来设置进度。使用拖动条控件 SeekBar 的基本流程如下。

第 1 步：在文件 main.xml 中插入一个按钮，具体代码如下所示。

```
<Button android:id="@+id/seek_bar_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="SeekBar"
/>
```

第 2 步：为按钮处理事件编写代码，对应代码如下所示。

```
private Button.OnClickListener seek_bar_button_listener = new Button.OnClickListener() {
        public void onClick(View v) {
            Intent intent = new Intent();
            intent.setClass(MainActivity.this, SeekBarActivity.class);
```

```
                startActivity(intent);
            }
        };
```

通过上述代码，当用户单击按钮后会跳转到 SeekBarActivty。

第 3 步：创建一个 Activty，为其指定模板 seek_bar.xml，文件 seek_bar.xml 的具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="wrap_content">
<TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="SeekBar" />
        <SeekBar
        android:id="@+id/seek"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:max="100"
        android:thumb="@drawable/seeker"
        android:progress="50"/>
</LinearLayout>
```

在上述代码中，定义了一个 SeekBar 控件，设置了其 ID 为 seek，设定了宽度为布满屏幕显示，设置了其最大值是 100。

第 4 步：在文件 AndroidManifest.xml 中增加对 SeekBarActivity 的申明，对应代码如下所示。

```
<activity android:name="SeekBarActivity" />
```

至此，整个实例设计完毕。执行后将显示对应样式的进度条，用户可以通过鼠标来拖动进度条的位置，如图 6-36 所示。

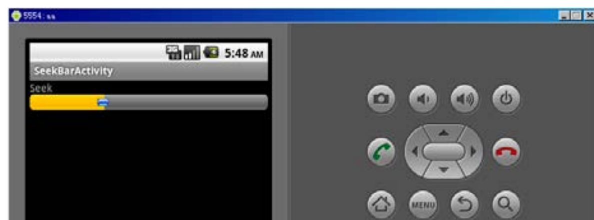


图 6-36　运行效果

6.3.15　使用评分组件RatingBar

评分组件 RatingBar 的功能是为用户提供一个评分操作的模式。在日常应用中，经常见到



评分系统，用户可以对某个产品或某个观点进行评分处理。使用评分组件 RatingBar 的基本流程如下所示。

第 1 步：在文件 main.xml 中插入一个按钮，具体代码如下所示。

```
<Button android:id="@+id/seek_bar_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="SeekBar"
/>
```

第 2 步：为按钮处理事件编写代码，对应代码如下所示。

```
private Button.OnClickListener rating_bar_button_listener = new Button.OnClickListener() {
        public void onClick(View v) {
            Intent intent = new Intent();
            intent.setClass(MainActivity.this, RatingBarActivity.class);
            startActivity(intent);
        }
    };
```

通过上述代码，当用户单击按钮后会跳转到 RatingBarActivty。

第 3 步：创建一个 Activty，为其指定模板 rating_bar.xml，文件 rating_bar.xml 的具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="wrap_content">
<TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="RatingBar"
/>
  <RatingBar android:id="@+id/rating_bar"
  android:layout_width="wrap_content"
  android:layout_height="wrap_content"
  ratingBarStyleSmall="true" />
</LinearLayout>
```

在上述代码中，定义了一个 RatingBar 控件，设置了其 ID 为 rating_bar，设定了宽度和高度都是自适应。

第 4 步：在文件 AndroidManifest.xml 中增加对 RatingBarActivity 的申明，对应代码如下所示。

```
<activity android:name="RatingBarActivity" />
```

至此，整个实例设计完毕。执行后将显示对应样式的评级图，用户可以通过鼠标来选择评级，如图 6-37 所示。

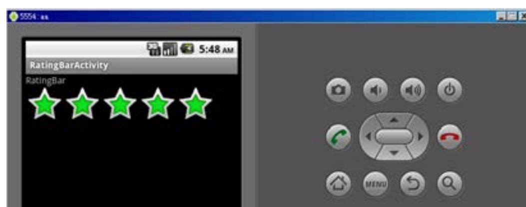


图 6-37　运行效果

6.3.16　使用图片视图控件ImageView

图片视图控件 ImageView 的功能是在屏幕中显示一张图片。使用图片视图控件 ImageView 的基本流程如下所示。

第 1 步：在文件 main.xml 中插入一个按钮，具体代码如下所示。

```
<Button android:id="@+id/image_view_button"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="ImageView"
/>
```

第 2 步：为按钮处理事件编写代码，对应代码如下所示。

```
private Button.OnClickListener image_view_button_listener = new Button.OnClickListener() {
            public void onClick(View v) {
                  Intent intent = new Intent();
                  intent.setClass(MainActivity.this, ImageViewActivity.class);
                  startActivity(intent);
            }
      };
```

通过上述代码，当用户单击按钮后会跳转到 ImageViewActivty 界面。

第 3 步：创建一个 Activty，为其指定模板 image_view.xml，文件 mage_view.xml 的具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical" android:layout_width="fill_parent"
      android:layout_height="wrap_content">
<TextView
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="图片展示:"
/>
<ImageView
   android:id="@+id/imagebutton"
   android:src="@drawable/eoe"
   android:layout_width="wrap_content"
```



```
    android:layout_height="wrap_content"/>
</LinearLayout>
```

在上述代码中，设置 Android:src 为一张图片，该图片位于本项目根目录下的“res\drawable”文件夹中，它支持 PNG、JPG、GIF 等常见的图片格式。

第 4 步：编写对应的 Java 程序，对应代码如下所示。

```
public class ImageViewActivity extends Activity {
        CheckBox plain_cb;
        CheckBox serif_cb;
        CheckBox italic_cb;
        CheckBox bold_cb;

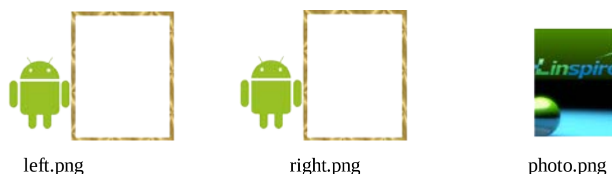
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                setTitle("ImageViewActivity");
                setContentView(R.layout.image_view);
//              find_and_modify_text_view();
        }
```

第 5 步：在文件 AndroidMainfest.xml 中增加对 ImageViewActivity 的声明，对应代码如下所示。

```
<activity android:name="ImageViewActivity" />
```

至此，整个实例设计完毕，执行后将显示对应的图片信息。

在实际项目应用中，通过 ImageView 可以对图片进行叠加处理。例如，想制作一个相框，需要使用如图 6-38 所示的 3 张素材图片。



left.png　　right.png　　photo.png

图 6-38　素材图片

具体实现流程如下。

第 1 步：修改 main.xml 布局，添加 UI 元素。具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
      android:id="@+id/widget34"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      xmlns:android="http://schemas.android.com/apk/res/android"
```

```
><!--创建第一个 ImageView (第二层图片)-->
<ImageView
    android:id="@+id/myImageView1"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="0px"
    android:layout_y="36px"
/>
<!--创建第二个 ImageView (第一层图片)-->
<ImageView
    android:id="@+id/myImageView2"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="0px"
    android:layout_y="36px"
/>
<!--创建第一个 Button -->
<Button
    android:id="@+id/myButton1"
    android:layout_width="105px"
    android:layout_height="66px"
    android:text="pic1"
    android:layout_x="9px"
    android:layout_y="356px"
/>
<!--创建第二个 Button -->
<Button
    android:id="@+id/myButton2"
    android:layout_width="105px"
    android:layout_height="66px"
    android:text="pic2"
    android:layout_x="179px"
    android:layout_y="356px"
/>
</AbsoluteLayout>
```

第 2 步：修改 mainActivity.java 文件，对应代码如下所示。

```
package zyf.Ex_Ctrl_7;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
public class Ex_Ctrl_7 extends Activity {
    /** Called when the activity is first created. */
    /* 声明 Button、ImageView 对象 */
    private ImageView mImageView01;
```



```
        private ImageView mImageView02;
        private Button mButton01;
        private Button mButton02;
        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.main);
            /* 取得 Button、ImageView 对象 */
            mImageView01 = (ImageView) findViewById(R.id.myImageView1);
            mImageView02 = (ImageView) findViewById(R.id.myImageView2);
            mButton01 = (Button) findViewById(R.id.myButton1);
            mButton02 = (Button) findViewById(R.id.myButton2);
            /* 设置 ImageView 背景图 */
            mImageView01.setImageDrawable(getResources().getDrawable(
                    R.drawable.right));
            mImageView02.setImageDrawable(getResources().getDrawable(
                    R.drawable.photo));
            /* 用 OnClickListener 事件来启动 */
            mButton01.setOnClickListener(new Button.OnClickListener() {
                @Override
                public void onClick(View v) {
                    /* 当启动后， ImageView 立刻换背景图 */
                    mImageView01.setImageDrawable(getResources().getDrawable(
                            R.drawable.right));
                }
            });
            mButton02.setOnClickListener(new Button.OnClickListener() {
                @Override
                public void onClick(View v) {
                    mImageView01.setImageDrawable(getResources().getDrawable(
                            R.drawable.left));
                }
            });
        }
    }
```

这样，即实现了对 3 张素材图片的叠加处理，程序执行后的效果如图 6-39 所示。



图 6-39　运行效果

6.3.17　使用图片按钮控件ImageButton

图片按钮控件 ImageButton 的功能是，在系统中将一张图片作为按钮来使用。通过使用 ImageButton，可以使项目中的按钮更加美观大方。使用图片按钮控件 ImageButton 的基本流程如下所示。

第 1 步：在文件 main.xml 中插入一个按钮，具体代码如下所示。

```
<Button android:id="@+id/image_button_button"
            android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:text="ImageButton" />
```

第 2 步：为按钮处理事件编写代码，对应代码如下所示。

```
private Button.OnClickListener image_button_button_listener = new Button.OnClickListener() {
            public void onClick(View v) {
                  Intent intent = new Intent();
                  intent.setClass(MainActivity.this, ImageButtonActivity.class);
                  startActivity(intent);
            }
      };
```

通过上述代码，当用户单击按钮后会跳转到 ImageButtonActivty。

第 3 步：为创建的 Activty 指定模板 image_button.xml，文件 image_button.xml 的具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical" android:layout_width="fill_parent"
      android:layout_height="wrap_content">
<TextView
          android:layout_width="wrap_content"
          android:layout_height="wrap_content"
          android:text="图片按钮:"
/>
<ImageButton id="@+id/imagebutton"
   android:src="@drawable/play"
   android:layout_width="wrap_content"
   android:layout_height="wrap_content"/>
</LinearLayout>
```

在上述代码中，设置了 Android:src 为一张图片，该图片位与本项目根目录下的“res\drawable”文件夹中，它支持 PNG、JPG、GIF 等常见的图片格式。

第 4 步：编写对应的 Java 程序，对应代码如下所示。

```
public class ImageButtonActivity extends Activity {
      CheckBox plain_cb;
```



```
        CheckBox serif_cb;
        CheckBox italic_cb;
        CheckBox bold_cb;
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setTitle("ImageButtonActivity");
            setContentView(R.layout.image_button);
//          find_and_modify_text_view();
        }
```

第 5 步：在文件 AndroidMainfest.xml 中增加对 ImageButtonActivity 的申明，对应代码如下所示。

```
    <activity android:name="ImageButtonActivity" />
```

至此，整个实例设计完毕。执行后将显示一个按钮，此按钮使用指定的图片实现的。具体效果如图 6-40 所示。

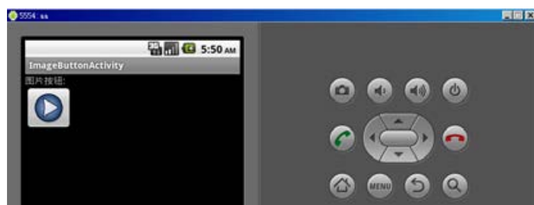


图 6-40　运行效果

6.3.18　使用切换图片控件ImageSwitcher和Gallery

切换图片控件有两个，分别是 ImageSwitcher 和 Gallery，它们的功能是以滑动的方式展现图片。在具体效果上，将首先显示一张大图，然后在大图下面显示一组可以滚动的小图。上述显示方式在现实中十分常见，如 QQ 空间的照片，如图 6-41 所示。

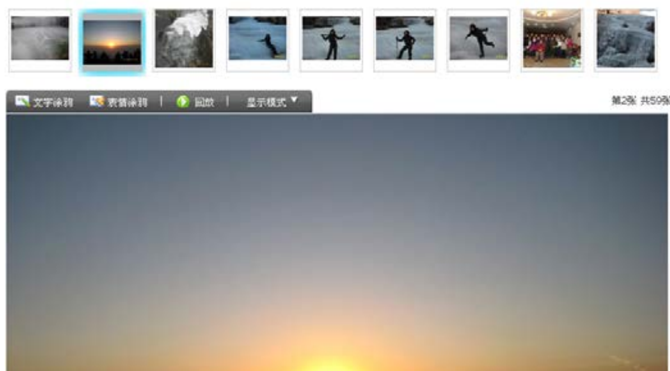


图 6-41　QQ 空间照片

使用切换图片控件 ImageSwitcher 和 Gallery 基本流程如下。

第 1 步：在文件 main.xml 中插入一个按钮，具体代码如下所示。

```
<Button android:id="@+id/image_show_button"
            android:layout_width="wrap_content"
             android:layout_height="wrap_content"
             android:text="ImageSwitcher Gallery"
/>
```

第 2 步：为按钮处理事件编写代码，对应代码如下所示。

```
private Button.OnClickListener image_show_button_listener = new Button.OnClickListener() {
             public void onClick(View v) {
                   Intent intent = new Intent();
                   intent.setClass(MainActivity.this, ImageShowActivity.class);
                   startActivity(intent);
             }
      };
```

通过上述代码，当用户单击按钮后会跳转到 ImageShowActivty。

第 3 步：为创建的 Activty 指定模板 image_show.xml，文件 image_button.xml 的具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout
         xmlns:android="http://schemas.android.com/apk/res/android"
         android:layout_width="fill_parent"
         android:layout_height="fill_parent"
>
<ImageSwitcher
         android:id="@+id/switcher"
                  android:layout_width="fill_parent"
                  android:layout_height="fill_parent"
                  android:layout_alignParentTop="true"
                  android:layout_alignParentLeft="true"
/>
<Gallery android:id="@+id/gallery"
         android:background="#55000000"
                  android:layout_width="fill_parent"
                  android:layout_height="60dp"
                  android:layout_alignParentBottom="true"
                  android:layout_alignParentLeft="true"
                  android:gravity="center_vertical"
                  android:spacing="16dp"
/>
</RelativeLayout>
```



在上述代码中，在 RelativeLayout 中插入了两个控件，分别是 ImageSwitcher 和 Gallery。其中 ImageSwitcher 用于显示上面那张大图，Gallery 用于控制下面小图列表索引。

第 4 步：编写对应的 Java 程序，首先编写 onCreate 函数，对应的代码如下所示。

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        requestWindowFeature(Window.FEATURE_NO_TITLE);
        setContentView(R.layout.image_show);
        setTitle("ImageShowActivity");

        mSwitcher = (ImageSwitcher) findViewById(R.id.switcher);
        mSwitcher.setFactory(this);
        mSwitcher.setInAnimation(AnimationUtils.loadAnimation(this,
        android.R.anim.fade_in));
        mSwitcher.setOutAnimation(AnimationUtils.loadAnimation(this,
        android.R.anim.fade_out));
        Gallery g = (Gallery) findViewById(R.id.gallery);
        g.setAdapter(new ImageAdapter(this));
        g.setOnItemSelectedListener(this);
    }
```

在上述代码中，通过使用 requestWindowFeature(Window.FEATURE_NO_TITLE) 设置 Activity 没有 titlebar，这样，此图片的显示区域就会增大。而类 Gallery 的使用方法和 ListView 差不多，也需要使用 setAdapter 来进行资源设置。

第 5 步：对 BaseAdapter 进行封装，通过 GetView 函数来返回要显示的 ImageView，GetView 函数的具体实现代码如下所示。

```
public View getView(int position, View convertView, ViewGroup parent) {
            ImageView i = new ImageView(mContext);
            i.setImageResource(mThumbIds[position]);
            i.setAdjustViewBounds(true);
            i.setLayoutParams(new Gallery.LayoutParams(
            LayoutParams.WRAP_CONTENT, LayoutParams.WRAP_CONTENT));
            i.setBackgroundResource(R.drawable.picture_frame);
            return i;
        }
```

在上述代码中，动态生成了一个 ImageView，然后分别使用 setImageResource、setLayoutParams 和 setBackgroundResource，分别实现了图片源文件、图片大小和图片背景的设置。当图片被显示到当前屏幕时，此函数会自动回调来提供要显示的 ImageView。

第 6 步：在 ImageSwitcher1 中实现 ViewSwitcher.ViewFactory 接口，在 ViewSwitcher.ViewFactory 接口中存在方法 View makeView，其实现代码如下所示。

```
public View makeView() {
        ImageView i = new ImageView(this);
```

```
            i.setBackgroundColor(0xFF000000);
            i.setScaleType(ImageView.ScaleType.FIT_CENTER);
            i.setLayoutParams(new ImageSwitcher.LayoutParams(LayoutParams.FILL_PARENT,
                    LayoutParams.FILL_PARENT));
            return i;
        }
```

通过上述代码，为 ImageSwitcher 返回了一个 View，在调用 ImageSwitcher 时，首先通过 Factory 为其提供一个 View，然后 ImageSwitcher 就可以初始化各种资源了。

第 7 步：在文件 AndroidMainfest.xml 中增加对 Activity 的申明，对应代码如下所示。

```
<activity android:name="ImageShowActivity" />
```

至此，整个实例设计完毕。执行后将会按照 QQ 空间中的照片样式显示，如图 6−42 所示。

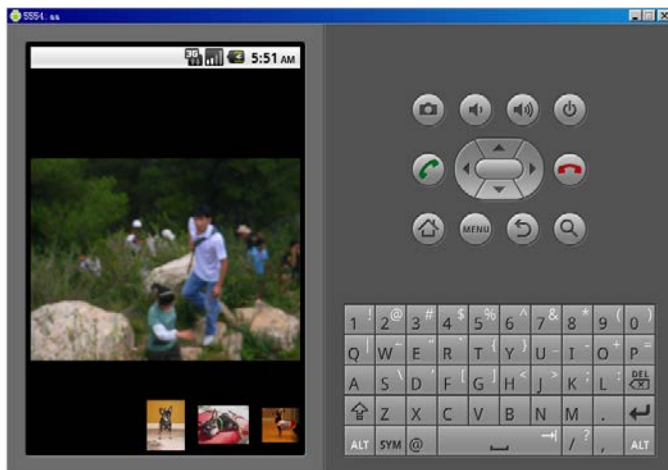


图 6−42　执行效果

6.3.19　使用网格视图控件GridView

网格视图控件 GridView 的功能是将很多张指定的图片以指定的大小显示出来。此功能在相册的图片浏览中比较常见。使用网格视图控件 GridView 的基本流程如下所示。

第 1 步：在文件 main.xml 中插入一个按钮，具体代码如下所示。

```
<Button android:id="@+id/grid_view_button"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="GridView"
/>
```

第 2 步：为按钮处理事件编写代码，对应代码如下所示。

```
private Button.OnClickListener grid_view_button_listener = new Button.OnClickListener() {
        public void onClick(View v) {
```



```
                Intent intent = new Intent();
                intent.setClass(MainActivity.this, GridViewActivity.class);
                startActivity(intent);
            }
        };
```

通过上述代码，当用户单击按钮后会跳转到 ImageShowActivty。

第 3 步：编写文件 GridViewActivity.java，首先创建 onCreate 方法，具体代码如下。

```
    public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.grid_view);
            setTitle("GridViewActivity");
            GridView gridview = (GridView) findViewById(R.id.grid_view);
            gridview.setAdapter(new ImageAdapter(this));
        }
```

在上述代码中，为创建的 Activty 指定模板 grid_view.xml，然后获取其模板中的 GridView 控件，并使用 setAdapter 方法为其绑定一个合适的 ImageAdapter，最后编写实现 ImageAdapter 的代码。

```
    public class ImageAdapter extends BaseAdapter {
            private Context mContext;
            public ImageAdapter(Context c) {
                mContext = c;
            }
            public int getCount() {
                return mThumbIds.length;
            }
            public Object getItem(int position) {
                return null;
            }
            public long getItemId(int position) {
                return 0;
            }
            public View getView(int position, View convertView, ViewGroup parent) {
                ImageView imageView;
                if (convertView == null) {    // if it's not recycled, initialize some attributes
                    imageView = new ImageView(mContext);
                    imageView.setLayoutParams(new GridView.LayoutParams(85, 85));
                    imageView.setScaleType(ImageView.ScaleType.CENTER_CROP);
                    imageView.setPadding(8, 8, 8, 8);
                } else {
                    imageView = (ImageView) convertView;
                }
                imageView.setImageResource(mThumbIds[position]);
                return imageView;
```

```
        }
        // references to our images
        private Integer[] mThumbIds = {
                R.drawable.grid_view_01, R.drawable.grid_view_02,
                R.drawable.grid_view_03, R.drawable.grid_view_04,
                R.drawable.grid_view_05, R.drawable.grid_view_06,
                R.drawable.grid_view_07, R.drawable.grid_view_08,
                R.drawable.grid_view_09, R.drawable.grid_view_10,
                R.drawable.grid_view_11, R.drawable.grid_view_12,
                R.drawable.grid_view_13, R.drawable.grid_view_14,
                R.drawable.grid_view_15, R.drawable.sample_1,
                R.drawable.sample_2, R.drawable.sample_3,
                R.drawable.sample_4, R.drawable.sample_5,
                R.drawable.sample_6, R.drawable.sample_7
        };
    }
```

在上述代码中，因为 ImageAdapter 继承于 BaseAdapter，所以可以通过构造方法 ImageAdapter 获取 Context。然后实现了 getView。

第 4 步：在文件 AndroidManifest.xml 中增加对 Activity 的申明，对应代码如下所示。

```
<activity android:name="GridViewActivity" />
```

至此，整个实例设计完毕。执行后将会按照格子视图的方式显示指定的图片，如图 6-43 所示。



图 6-43　运行效果

6.3.20　使用标签控件Tab

标签控件 Tab 的功能是在屏幕内实现多个标签栏样式的效果，当单击某个标签栏时，会打开对应的界面。使用标签控件 Tab 的基本流程如下。

第 1 步：在文件 main.xml 中插入一个按钮，具体代码如下所示。

```
<Button android:id="@+id/tab_demo_button"
        android:layout_width="wrap_content"
```



```
            android:layout_height="wrap_content"
            android:text="TabView"
    />
```

第 2 步：为按钮处理事件编写代码，对应代码如下所示。

```
    private Button.OnClickListener tab_demo_button_listener = new Button.OnClickListener() {
            public void onClick(View v) {
                Intent intent = new Intent();
                intent.setClass(MainActivity.this, TabDemoActivity.class);
                startActivity(intent);
            }
        };
```

第 3 步：编写 Java 文件 TabDemoActivity.java，用于集成 TabActivity，完成 TabDemo-Activity 的实现，具体代码如下。

```
    @Override
        public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setTitle("TabDemoActivity");
            TabHost tabHost = getTabHost();
            LayoutInflater.from(this).inflate(R.layout.tab_demo,
                    tabHost.getTabContentView(), true);
            tabHost.addTab(tabHost.newTabSpec("tab1").setIndicator("tab1")
                    .setContent(R.id.view1));
            tabHost.addTab(tabHost.newTabSpec("tab3").setIndicator("tab2")
                    .setContent(R.id.view2));
            tabHost.addTab(tabHost.newTabSpec("tab3").setIndicator("tab3")
                    .setContent(R.id.view3));
        }
```

在上述代码中，首先通过 getTabContentView 获取了 TabHost，然后绑定其模板，这样就绑定了每个标签的内容关联。

第 4 步：编写模板文件 tab_demo.xml，具体代码如下所示。

```
    <?xml version="1.0" encoding="utf-8"?>
    <FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">

        <TextView android:id="@+id/view1"
            android:background="@drawable/blue"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:text="这里是 Tab1 里的内容。"/>
        <TextView android:id="@+id/view2"
```

```
            android:background="@drawable/red"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:text="这里是 Tab2，balabalal....。"/>

        <TextView android:id="@+id/view3"
            android:background="@drawable/green"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:text="Tab3"/>

    </FrameLayout>
```

通过上述代码，在里面插入了 3 个 TextView 控件，当每个标签切换的时候，会显示各自对应的 TextView。

第 5 步：在文件 AndroidManifest.xml 中增加对 Activity 的声明，对应代码如下所示。

```
    <activity android:name="TabDemoActivity" />
```

至此，整个实例设计完毕，执行后将会按指定的样式显示对应标签，如图 6-44 所示。



图 6-44　运行效果

到此为止，Android 中 Widget 组件的基本知识介绍完毕，读者可以参考本书光盘中的源代码，仔细阅读本书，体味每个知识点和实例结合运用的方法。

6.4　使用友好菜单控件Menu

控件 Menu 的功能是为用户提供一个友好的界面显示效果。在本节的内容中，将详细介绍控件 Menu 的基本知识，并详细讲解创建控件 Menu 的具体过程。

6.4.1　Menu基础知识介绍

大部分的应用程序都包括两种人机互动方式，一种是直接通过 GUI 的 Views，其可以满足大部分的交互操作；另外一种是应用 Menu，当按下“Menu”按钮后，会弹出与当前活动



状态下的应用程序相匹配的菜单。这两种方式相比较都有各自的优势，而且可以很好地相辅相成，即便用户可以由主界面完成大部分操作，但是适当地拓展 Menu 功能可以更加完善应用程序，至少用户可以通过排列整齐的按钮清晰地了解当前模式下可以使用的功能。

Android 提供了 3 种菜单类型，分别为 Options Menu，Context Menu，Sub Menu。

options menu 就是通过按〈home〉键来显示，Context Menu 需要在 View 上按上 2s 后显示。这两种 Menu 都可以加入子菜单，子菜单不能嵌套子菜单。Options Menu 最多只能在屏幕最下面显示 6 个菜单选项，称为 Icon Menu，Icon Menu 不能有 Checkable 选项。多于 6 的菜单项会以 More Icon Menu 来调出，称为 Expanded Menu。Options Menu 通过 Activity 的 onCreateOptionsMenu 来生成，这个函数只会在 menu 第一次生成时调用。任何想改变 Options Menu 的想法只能在 onPrepareOptionsMenu 来实现，这个函数会在 Menu 显示前调用。onOptionsItemSelected 用来处理选中的菜单项。

Context Menu 是跟某个具体的 View 绑定在一起的，在 Activity 中用 Register ForContextMenu 来为某个 View 注册 Context Menu。Context Menu 在显示前都会调用 onCreateContextMenu 来生成 Menu。onContextItemSelected 用来处理选中的菜单项。

Android 还提供了对菜单项进行分组的功能，可以把相似功能的菜单项分在同一个组，这样就可以通过调用 setGroupCheckable，setGroupEnabled,setGroupVisible 来设置菜单属性，而无须单独设置。

6.4.2　使用Menu实例

在本节的内容中，将通过一个实例的实现过程，来讲解使用 Menu 控件的基本流程。本实例保存在“光盘:\6\shiyongmenu”，具体实现流程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“shiyongmenu”的工程文件，如图 6-45 所示。

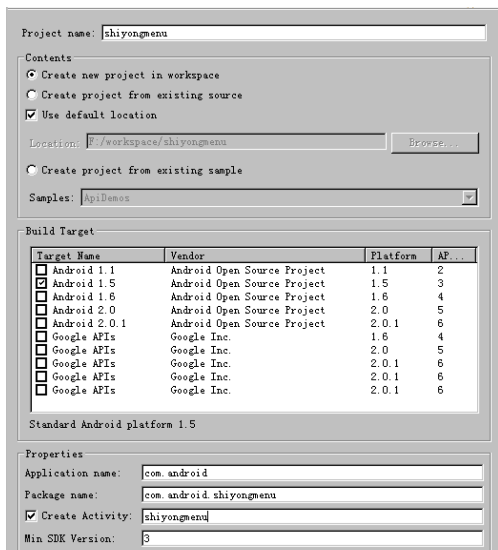


图 6-45　创建一个工程

第 2 步：编写 main.xml 主文件，此文件是一个布局文件，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="@string/hello" />
    <Button android:id="@+id/button1"
        android:layout_width="100px"
        android:layout_height="wrap_content" android:text="@string/button1" />
    <Button android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="@string/button2" />
</LinearLayout>
```

通过上述代码，插入了 1 个 TextView 控件和两个 Button 控件。其中 TextView 用于显示文本，然后用“layout_width”设置了 Button 的宽度，用“layout_height”设置了 Button 的高度；最后，通过符号@来设置了读取变量值，并进行替换，具体说明如下。

- android:text="@string/button1"：相当于<string name="button1">button1</string>。
- android:text="@string/button2"：相当于<string name="button2">button2</string>。

大家不能小看上面的符号@，它用于提示 XML 文件的解析器要对@后面的名字进行解析，例如，上面的"@string/button1"，解析器会从 values/string.xml 中读取 Button1 这个变量值。

文件 string.xml 中定义了 TextView 和 Button 的值，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello World, ActivityMenu</string>
    <string name="app_name">HelloMenu</string>
    <string name="button1">button1</string>
    <string name="button2">button2</string>
</resources>
```

第 3 步：编写 Java 文件 ActivityMenu.java，其实现流程如下。

1）定义函数 onCreate，先用于显示出 main.xml 描述的 Layout，并设置两个 Button 为不可见状态。

2）定义函数 onCreateOptionsMenu，用于生成 Menu，此函数是一个回调方法，只有当按下手机设备上的 Menu 按钮后，Android 才会生成一个包含两个子项的菜单。在具体实现上，将首先得到 super 函数调用后的返回值，并在 onCreateOptionsMenu 的最后返回；然后调用 menu.add 给 Menu 添加一个项。

3）定义函数 onOptionsItemSelected，此函数是一个回调方法，只有当按下手机设备上的 Menu 按钮后，Android 才会调用执行。而这个事件就是单击菜单里的某一项，即 MenuItem。

主要代码如下所示。



```
public class ActivityMenu extends Activity {
    public static final int ITEM0 = Menu.FIRST;
    public static final int ITEM1 = Menu.FIRST + 1;
    Button button1;
    Button button2;

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState) {
       super.onCreate(savedInstanceState);
       setContentView(R.layout.main);
       button1 = (Button) findViewById(R.id.button1);
       button2 = (Button) findViewById(R.id.button2);
       /*  设置 2 个 button 不可见  */
       button1.setVisibility(View.INVISIBLE);
       button2.setVisibility(View.INVISIBLE);
           }
    @Override
    /*
     * menu.findItem(EXIT_ID);找到特定的 MenuItem
     * MenuItem.setIcon.可以设置 menu 按钮的背景
     */
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        menu.add(0, ITEM0, 0, "显示 button1");
        menu.add(0, ITEM1, 0, "显示 button2");
        menu.findItem(ITEM1);
        return true;
    }

    public boolean onOptionsItemSelected(MenuItem item) {
        switch (item.getItemId()) {
        case ITEM0:
            actionClickMenuItem1();
        break;
        case ITEM1:
            actionClickMenuItem2(); break;

        }
        return super.onOptionsItemSelected(item);}
    /*
     *  点击第一个 menu 的第一个按钮执行的动作
     */
    private void actionClickMenuItem1(){
        setTitle("button1  可见");
        button1.setVisibility(View.VISIBLE);
        button2.setVisibility(View.INVISIBLE);
    }
    /*
     *  点击第二个 Menu 的第一个按钮执行的动作
```

```
         */
        private void actionClickMenuItem2(){
                setTitle("button2  可见");
                button1.setVisibility(View.INVISIBLE);
                button2.setVisibility(View.VISIBLE);
        }
}
```

至此，整个项目实例介绍完毕，执行后的效果如图 6-46 所示。当单击设备上的“MENU”键后会触发程序，并在屏幕中显示预先设置的已经隐藏的 2 个按钮，如图 6-47 所示。

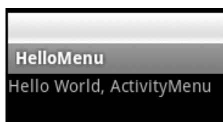
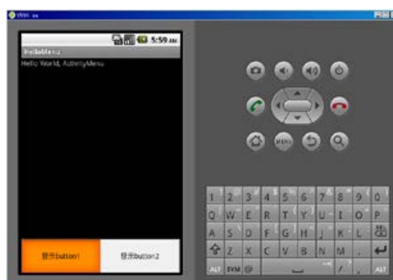


图 6-46　初始效果


图 6-47　触发设备后的效果

6.5　Intent和Activity

由前面的内容可知，Activity 是 Android 中最重要的知识点之一，也是最常用的。关于对 Activity 的使用，一直是 Android 领域的重点。本章前面的实例，都是基于一个 Activity，实际上我们可以一次使用多个 Activity，并且这些 Activity 之间是可以相互跳转的，相互之间可以传递数据。在本节的内容中，将通过一个简单实例的实现过程，来讲解 Intent 和 Activity 联合使用的过程，并在过程中穿插讲解两者的使用技巧。本实例保存在“光盘:\daima\6\”中，下面开始讲解本实例的具体实现流程。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“activity_intent”的工程文件，如图 6-45 所示。

第 2 步：编写 main.xml 主文件，此文件是一个布局文件，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical" android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <Button android:id="@+id/button1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" android:text="button1" />
        <Button android:id="@+id/button2"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content" android:text="button2" />
</LinearLayout>
```



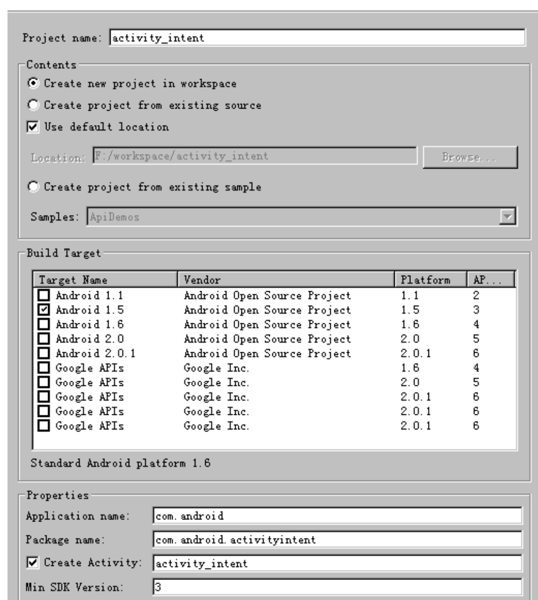


图 6-48　创建一个工程

通过上述代码，插入了 2 个 Button 控件。

第 3 步：编写主程序 ActivityMain.java，下面开始讲解主程序 ActivityMain.java 的具体实现过程。

1）首先在里面创建 onCreat 函数，具体代码如下所示。

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        listener1 = new OnClickListener() {
            public void onClick(View v) {
                Intent intent1 = new Intent(ActivityMain.this, Activity1.class);
                intent1.putExtra("activityMain", "来自 activityMain");
                startActivityForResult(intent1, REQUEST_CODE);
            }
        };
```

在上述代码中，首先把 layout 目录中 main.xml 中的 layout 显示出来，然后才能得到 button 的应用，依次绑定一个单击监听器“OnClickListener”。

2）编写第一个跳转代码处理程序，具体代码如下。

```
listener1 = new OnClickListener() {
            public void onClick(View v) {
                Intent intent1 = new Intent(ActivityMain.this, Activity1.class);
                intent1.putExtra("activityMain", "来自 activityMain");
                startActivityForResult(intent1, REQUEST_CODE);
```

在上述代码中，实现了当单击 button1 时，程序从 ActivityMain 跳转到 Activity1。

3）编写第二个跳转代码处理程序，具体代码如下。

```
        listener2 = new OnClickListener() {
                        public void onClick(View v) {
                            setTitle("在 ActivityMain");
                            Intent intent2 = new Intent(ActivityMain.this, Activity2.class);
                            startActivity(intent2);
```

在上述代码中，实现了当单击 button2 时，程序跳转到 Activity2。即在单击 button2，系统反向调用绑定在 button2 上的监听器的 onClick 方法。

4）编写函数 onActivityResult，此函数用于启动 Intent，并在新的 Activity 运行完毕后，执行原来 Activity 中的回调函数，此回调函数是 onActivityResult，对应代码如下。

```
    @Override
        protected void onActivityResult(int requestCode, int resultCode, Intent data) {
            if (requestCode == REQUEST_CODE) {
                if (resultCode == RESULT_CANCELED)
                    setTitle("取消");
                else if (resultCode == RESULT_OK) {
                    String temp=null;
                     Bundle extras = data.getExtras();
                        if (extras != null) {
                          temp = extras.getString("store");
                        }
                    setTitle(temp);
                }
            }
        }
```

第 4 步：编写文件 Activity2.java，使用一个新的 Activity—Activity2，并实现它和 activity2.xml 的关联，具体代码如下所示。

```
public class Activity2 extends Activity {
    OnClickListener listener = null;
    Button button;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity2);
        listener = new OnClickListener() {
            public void onClick(View v) {
                finish();
            }
        };
        button = (Button) findViewById(R.id.button4);
        button.setOnClickListener(listener);
        setTitle("在 Activity2");
    }
}
```



第 5 步：编写文件 Activity1.java，具体实现流程如下。

1）单击 button1 后 Activity 实现跳转，跳转到了 Activity1。在 Activity1 中，程序会把 ActivityMain 传递来的值显示在屏幕标题 title 中，具体实现代码如下。

```
Bundle extras = getIntent().getExtras();
        if (extras != null) {
            data = extras.getString("activityMain");
        }
    setTitle("在 Activity1:"+data);
```

2）单击 Activity1 中的 button3 后，此 Activity 结束，此时会将需要返回的数据放置到 Intent 中，对应代码如下所示。

```
listener1 = new OnClickListener() {
        public void onClick(View v) {
            Bundle bundle = new Bundle();
            bundle.putString("store", "来自 Activity1");
            Intent mIntent = new Intent();
            mIntent.putExtras(bundle);
            setResult(RESULT_OK, mIntent);
            finish();
        }
    };
```

第 6 步：编写文件 AndroidManifest.xml，加入新的 Activity，对应代码如下所示。

```
<activity android:name=".ActivityMain" android:label="@string/app_name">
-<intent-filter>
<action android:name="android.intent.action.MAIN" />
<category android:name="android.intent.category.LAUNCHER" />
</intent-filter>
</activity>
<!—下面是新加的 Activity-->
<activity android:name=".Activity1" />
<activity android:name=".Activity2" />
```

至此，整个项目实例介绍完毕，执行后的效果如图 6-49 所示；单击“button1”后的效果如图 6-50 所示；单击“button3”后的效果如图 6-51 所示；单击“button2”后的效果如图 6-52 所示。

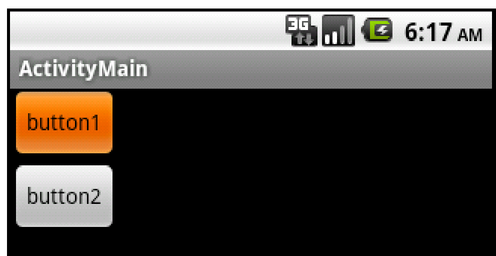


图 6-49　初始效果



图 6-50　单击“button1”后的效果

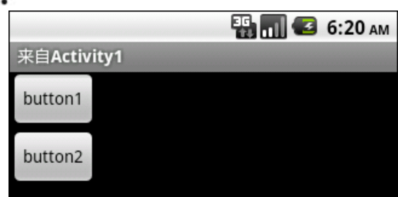


图 6-51　单击“button3”后的效果

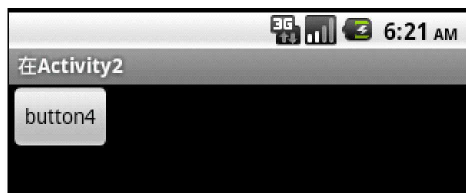


图 6-52　单击“button2”后的效果

6.6　使用列表控件ListView

ListView 是 Android 开发中最常用的组件之一，能够在屏幕内实现列表显示。ListView 通过一个 Adapter 来构建显示的，在 Android 中可以使用 3 种 Adapter，分别是 ArrayAdapter、SimpleAdapter、CursorAdapter。本节详细讲解 ListView 的基本使用方法。

6.6.1　ArrayAdapter接受一个数组或者列表（List）作为参数来构建

ArrayAdapter 可以接受一个数组或者列表（List）作为参数，来构建数据并显示。下面通过一个简单例子说明实现过程，先创建 Test 继承 ListActivity（活动列表），这里传入一个 string 数组，具体代码如下。

```
public class ListTest extends ListActivity {
    /** *//** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        String[] sw = new String[100];
        for (int i = 0; i < 100; i++) {
            sw[i] = "listtest_" + i;
        }
        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,android.R.layout.simple_list
_item_1,sw);          //使用系统已经实现好的 xml 文件 simple_list_item_1
        setListAdapter(adapter);
    }
}
```

程序执行后效果如图 6-53 所示。

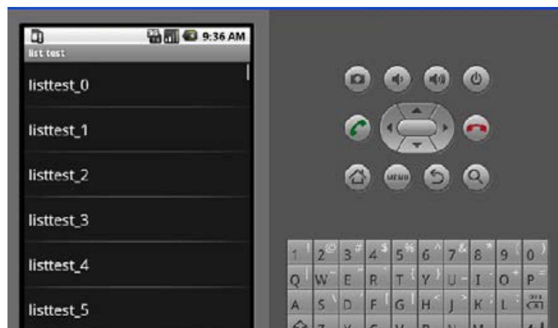


图 6-53　运行效果



从以上代码的执行效果可以看出，不需要加载自己的 Layout（布局）而只用系统已经实现的 Layout 就能很快实现 Listview 效果功能。

6.6.2　使用SimpleAdapter

下面通过一个实例来说明使用 SimpleAdapter 实现 ListView 的方法，本实例保存在“光盘:\daima\6\my_list”，具体实现流程如下。

第 1 步：构建 List 列表，并设置每项为一个 Map 来实现。编写代码，创建 TestList（测试列表）类继承 Activity。具体代码如下所示。

```
super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        ArrayList<HashMap<String, Object>> users = new ArrayList<HashMap<String, Object>>();
        for (int i = 0; i < 10; i++) {
            HashMap<String, Object> user = new HashMap<String, Object>();
            user.put("img", R.drawable.user);
            user.put("username", "姓名(" + i+")");
            user.put("age", (20 + i) + "");
            users.add(user);
        }
        SimpleAdapter saImageItems = new SimpleAdapter(this,
                users,// 数据来源
                R.layout.user,//每一个 user xml 相当 ListView 的一个组件
                new String[] { "img", "username", "age" },
                // 分别对应 view 的 id
                new int[] { R.id.img, R.id.name, R.id.age });
        // 获取 listview
((ListView) findViewById(R.id.users)).setAdapter(saImageItems);
```

第 2 步：编写文件 main.xml 实现布局，插入 3 个 TextView，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:text="用户列表" android:gravity="center"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent" android:background="#DAA520"
        android:textColor="#000000">
    </TextView>
    <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content">
        <TextView android:text="姓名"
            android:gravity="center" android:layout_width="160px"
            android:layout_height="wrap_content" android:textStyle="bold"
            android:background="#7CFC00">
```

```
            </TextView>
            <TextView android:text="年龄"
                android:layout_width="170px" android:gravity="center"
                android:layout_height="wrap_content" android:textStyle="bold"
                android:background="#F0E68C">
            </TextView>
        </LinearLayout>
    <ListView android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:id="@+id/users">
        </ListView>
    </LinearLayout>
```

在上述代码中，ListView 前面的是标题行，ListView 相当于用来显示数据的容器，里面每行是一个用户信息。

第 3 步：编写文件 use.xml，用于定义用户信息布局，具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout
        android:layout_width="fill_parent"
        xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_height="wrap_content"
        >
        <TableRow >
        <ImageView
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:id="@+id/img">
        </ImageView>
        <TextView
            android:layout_height="wrap_content"
            android:layout_width="150px"
            android:id="@+id/name">
        </TextView>
        <TextView
            android:layout_height="wrap_content"
            android:layout_width="170px"
            android:id="@+id/age">
        </TextView>
        </TableRow>
</TableLayout>
```

通过上述代码，设置每行包含了一个 img（图片）和两段 TextViow（文字）信息，这个文件以参数的形式通过 Adapter 在 ListView 中显示。

第 4 步：编写主处理文件 ListTest.java，通过 SimpleAdapter 来显示每块区域的用户信息，具体代码如下所示。



```
        SimpleAdapter saImageItems = new SimpleAdapter(this,
                        users,                          // 数据来源
                        R.layout.user,                  //每一个 user xml 相当 ListView 的一个组件
                        new String[] { "img", "username", "age" },
                        // 分别对应 view 的 id
                        new int[] { R.id.img, R.id.name, R.id.age });
```

至此，整个实例介绍完毕，执行后的效果如图 6-54 所示。

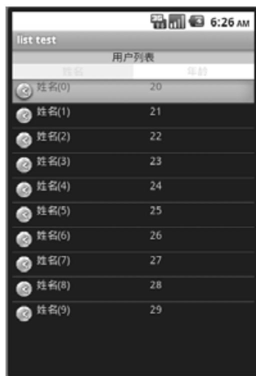


图 6-54　运行效果

6.7　使用互动对话框控件Dialog

对话框控件 Dialog 的功能是，在手机屏幕中实现互动对话框效果。本节将通过一个具体实例的实现过程，来讲解使用 Dialog 控件的方法。本实例保存在“光盘:\daima\6\shiyongdialog”。下面开始介绍本实例的实现过程，具体过程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“shiyongdialog”的工程文件，如图 6-55 所示。

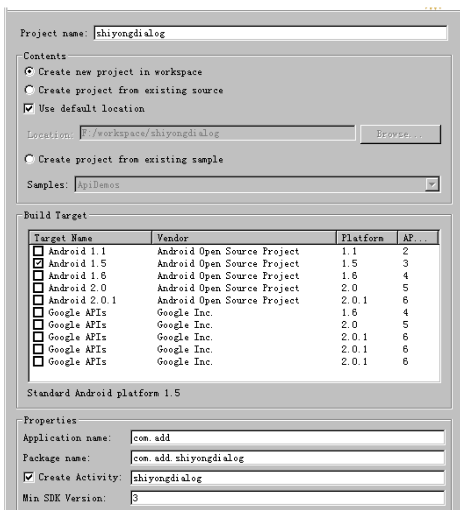


图 6-55　新建一个项目

第 2 步：编写主布局文件 main.xml，插入 layout 布局，其主要代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

第 3 步：开始编写程序文件 ActivityMain.java，具体实现过程如下。

1）首先看里面的 onCreateDialog 方法，具体代码如下所示。

```
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.alert_dialog);

    Button button1 = (Button) findViewById(R.id.button1);
    button1.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            showDialog(DIALOG1);
        }
    });

    Button buttons2 = (Button) findViewById(R.id.buttons2);
    buttons2.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            showDialog(DIALOG2);
        }
    });

    Button button3 = (Button) findViewById(R.id.button3);
    button3.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            showDialog(DIALOG3);
        }
    });

    Button button4 = (Button) findViewById(R.id.button4);
    button4.setOnClickListener(new OnClickListener() {
        public void onClick(View v) {
            showDialog(DIALOG4);
```



```
            }
        });
    }
```

上述代码的具体说明如下。

- 方法 findViewById 通过组件的 ID 返回对这个组件的引用。
- 方法 setOnClickListener 为 button1 设置了一个单击监听器。
- onClick 为单击 Button 后的回调函数。
- showDialog 是 Activity 里的函数，用于将 ID 为 DIALOG1 的 Dialog 显示出来。

2）方法 onCreateDialog，是一个回调函数，具体代码如下。

```
    @Override
        protected Dialog onCreateDialog(int id) {
            switch (id) {
            case DIALOG1:
                return buildDialog1(ActivityMain.this);

            case DIALOG2:
                return buildDialog2(ActivityMain.this);

            case DIALOG3:
                return buildDialog3(ActivityMain.this);
            case DIALOG4:
                return buildDialog4(ActivityMain.this);

            }
            return null;
        }
```

通过上述代码，能够根据不同的 Dialog 的 ID，生成不同的 Dialog，例如 buildDialog1 函数用于生成第一个要显示的 Dialog。

3）实现函数 buildDialog1、buildDialog2、buildDialog3 和 buildDialog4，具体实现代码如下。

```
    private Dialog buildDialog1(Context context) {
            /*创建一个 AlertDialog.Builder builder 对象*/
            AlertDialog.Builder builder = new AlertDialog.Builder(context);
            /*给 AlertDialog 预设一个图片*/
            builder.setIcon(R.drawable.alert_dialog_icon);
            /*给 AlertDialog 预设一个标题*/
            builder.setTitle(R.string.alert_dialog_two_buttons_title);
            /*设置按钮的属性*/
            builder.setPositiveButton(R.string.alert_dialog_ok,
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int whichButton) {
```

```
                        setTitle("这是点击了对话框上的确定按钮");
                    }
                });
        builder.setNegativeButton(R.string.alert_dialog_cancel,
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton) {

                        setTitle("这是点击了对话框上的取消按钮");
                    }
                });
        return builder.create();

    }
    private Dialog buildDialog2(Context context) {
        AlertDialog.Builder builder = new AlertDialog.Builder(context);
        builder.setIcon(R.drawable.alert_dialog_icon);
        builder.setTitle(R.string.alert_dialog_two_buttons_msg);
        builder.setMessage(R.string.alert_dialog_two_buttons2_msg);
        builder.setPositiveButton(R.string.alert_dialog_ok,
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton) {

                        setTitle("这是点击了对话框上的确定按钮");
                    }
                });
        builder.setNeutralButton(R.string.alert_dialog_something,
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton) {

                        setTitle("这是点击了对话框上的进入详细按钮");
                    }
                });
        builder.setNegativeButton(R.string.alert_dialog_cancel,
                new DialogInterface.OnClickListener() {
                    public void onClick(DialogInterface dialog, int whichButton) {

                        setTitle("这是点击了对话框上的取消按钮");
                    }
                });
        return builder.create();
    }

    private Dialog buildDialog3(Context context) {
        LayoutInflater inflater = LayoutInflater.from(this);
        final View textEntryView = inflater.inflate(
                R.layout.alert_dialog_text_entry, null);
```



```
            AlertDialog.Builder builder = new AlertDialog.Builder(context);
            builder.setIcon(R.drawable.alert_dialog_icon);
            builder.setTitle(R.string.alert_dialog_text_entry);
            builder.setView(textEntryView);
            builder.setPositiveButton(R.string.alert_dialog_ok,
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int whichButton) {
                            setTitle("这是点击了对话框上的确定按钮");
                        }
                    });
            builder.setNegativeButton(R.string.alert_dialog_cancel,
                    new DialogInterface.OnClickListener() {
                        public void onClick(DialogInterface dialog, int whichButton) {
                            setTitle("这是点击了对话框上的取消按钮");
                        }
                    });
            return builder.create();
        }

        private Dialog buildDialog4(Context context) {
            ProgressDialog dialog = new ProgressDialog(context);
            dialog.setTitle("正在处理");
            dialog.setMessage("请等会……");
            return  dialog;
        }
    }
```

上述 4 个函数的实现原理是一样的，下面开始讲解它们的实现过程。

- buildDialog1：onClick 方法是监听器中的回调方法，当单击 Dialog 按钮时，系统会回调这个方法；setNeutralButton 方法和 setPositiveButton 方法相对应，主要用于设置、取消按钮的一些属性；执行 builder.create 后，会生成一个配置好的 Dialog。
- buildDialog2：当单击第二个 button 后，会执行 buildDialog2，其中方法 setNeutralButton 用于设置中间按钮中的一些属性，具体设置方法和 buildDialog1 中的一样。
- buildDialog3：当单击第三个 button 后，会执行 buildDialog3，其中通过 LayoutInflater 类的 inflater 方法，可以将一个 XML 布局变为一个 View 实例。另外，如下语句是整个 Dialog 的精髓。

```
builder.setView(textEntryView);
```

通过上述方法可以将实现好的个性化的 View 放置到 Dialog 中去，此处的 textEntryView 和 alert_dialog_entry.xml 定义的布局相关联。

- buildDialog4：此程序最为简单，执行后将会显示一个等待界面。

第 4 步：编写文件 alert_dialog_text_entry.xml，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="wrap_content"
    android:orientation="vertical">
    <TextView android:id="@+id/username_view"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content" android:layout_marginLeft="20dip"
        android:layout_marginRight="20dip" android:text="输入用户名"
        android:textAppearance="?android:attr/textAppearanceMedium" />
    <EditText android:id="@+id/username_edit"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent" android:layout_marginLeft="20dip"
        android:layout_marginRight="20dip" android:capitalize="none"
        android:textAppearance="?android:attr/textAppearanceMedium" />
    <TextView android:id="@+id/password_view"
        android:layout_height="wrap_content"
        android:layout_width="wrap_content" android:layout_marginLeft="20dip"
        android:layout_marginRight="20dip" android:text="输入你的密码"
        android:textAppearance="?android:attr/textAppearanceMedium" />
    <EditText android:id="@+id/password_edit"
        android:layout_height="wrap_content"
        android:layout_width="fill_parent" android:layout_marginLeft="20dip"
        android:layout_marginRight="20dip" android:capitalize="none"
        android:password="true"
        android:textAppearance="?android:attr/textAppearanceMedium" />
</LinearLayout>
```

其中，“android:textAppearance="?android:attr/textAppearanceMedium"”用于设置 TextView 里面文字的字体；“EditText”是一个文本输入框；“android:password="true"”将输入框设置为密码输入框，只显示星号；在“android:capitalize="none"”中，通过 capitalize 属性设置输入字符的大小写，none 表示首字母小写。

第 5 步：编写文件 alert_dialog.xml，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/screen" android:layout_width="fill_parent"
    android:layout_height="fill_parent" android:orientation="vertical">
    <LinearLayout android:layout_width="fill_parent"
        android:layout_height="fill_parent" android:orientation="vertical">
        <Button android:id="@+id/button1"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:text="我是拥有两个 button 的对话框 1" />
        <Button android:id="@+id/buttons2"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:text="我是拥有三个 button 的对话框 2" />
        <Button android:id="@+id/button3"
```



```
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content" android:text="在这里能进行输入的对话框" />
                <Button android:id="@+id/button4"
                    android:layout_width="fill_parent"
                    android:layout_height="wrap_content" android:text="这是进度框，嘿嘿！ " />
            </LinearLayout>
        </ScrollView>
```

通过上述代码，设置了各个按钮上显示的文本。

至此，整个实例设计完毕。执行后的初始效果如图 6-56 所示；单击第一个 button 后的效果如图 6-57 所示；单击第二个 button 后的效果如图 6-58 所示；单击第三个 button 后的效果如图 6-59 所示；单击第四个 button 后的效果如图 6-60 所示。



图 6-56　初始效果



图 6-57　单击第一个 button 后的效果



图 6-58　单击第二个 button 后的效果



图 6-59　单击第三个 button 后的效果



图 6-60　单击第四个 button 后的效果

6.8　使用Toast和Notification

前面介绍的 Dialog 实际上已经实现了提醒功能，在 Android 中，还可以通过 Toast（提醒）和 Notification（通知）来实现提醒功能。和 Dialog 相比，这种提醒更加友好，并且不会打断

用户的当前操作。本节详细讲解 Toast 和 Notification 控件的具体使用方法。

6.8.1 Toast简介

Toast 是 Android 中用来显示信息的一种机制，和 Dialog 不一样的是，Toast 是没有焦点的，而且 Toast 显示的时间有限，过一定的时间就会自动消失。例如，在下面的代码中，编写了 Activity 的子类别 ToastDemo。

```
package com.a3gs.toast;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
public class ToastDemo extends Activity {
    private EditText myET;
    private Button myBtn;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
         myET = (EditText) findViewById(R.id.myET);
        myBtn = (Button) findViewById(R.id.myBtn);
        myBtn.setOnClickListener(new Button.OnClickListener(){
            @Override
            public void onClick(View v) {
                // TODO Auto-generated method stub
                Toast.makeText(ToastDemo.this, "您所填的信息是：" +
                          myET.getText ().toString(), Toast.LENGTH_LONG).show();
                myET.setText("");
            }
        });
    }
}
```

然后编写 main.xml 文件，其代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
```



```
        android:layout_height="wrap_content"
        android:text="@string/myText"
        />
    <EditText
        android:id="@+id/myET"
        android:layout_width="180px"
        android:layout_height="wrap_content"
        />
    <Button
        android:id="@+id/myBtn"
        android:layout_width="100px"
        android:layout_height="wrap_content"
        android:text="@string/BtnText"
        />
    </LinearLayout>
```

这样，就简单使用了 Toast 实现了提示功能。执行后的初始效果如图 6-61 所示；输入信息并单击“发送”按钮后，会以提示的方式显示输入的数据，如图 6-62 所示。

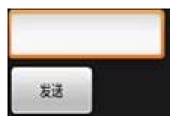


图 6-61　初始效果



图 6-62　输出的提醒信息

6.8.2　Notification简介

Notification 看名字就知道，是一个和提醒有关的东西，它通常和 NotificationManager 一块使用。具体来说，其主要功能如下。

1．NotificationManager和Notification用来设置通知

通知的设置等操作相对比较简单，基本的使用方式就是新建一个 Notification 对象，设置好通知的各项参数，然后使用系统后台运行的 NotificationManager 服务将通知发出来。基本步骤如下。

1）得到 NotificationManager，代码如下。

```
String ns = Context.NOTIFICATION_SERVICE;
NotificationManager mNotificationManager = (NotificationManager) getSystemService(ns);
```

2）创建一个新的 Notification 对象，代码如下。

```
Notification notification = new Notification();
notification.icon = R.drawable.notification_icon;
```

也可以使用稍微复杂一些的方式创建 Notification，代码如下。

```
int icon = R.drawable.notification_icon;                    //通知图标
CharSequence tickerText = "Hello";                          //状态栏(Status Bar)显示的通知文本提示
long when = System.currentTimeMillis();                     //通知产生的时间，会在通知信息里显示
Notification notification = new Notification(icon, tickerText, when);
```

3）填充 Notification 的各个属性，代码如下。

```
Context context = getApplicationContext();
CharSequence contentTitle = "My notification";
CharSequence contentText = "Hello World!";
Intent notificationIntent = new Intent(this, MyClass.class);
PendingIntent contentIntent = PendingIntent.getActivity(this, 0, notificationIntent, 0);
notification.setLatestEventInfo(context, contentTitle, contentText, contentIntent);
```

Notification 提供了如下几种手机提示方式。

- 在状态栏（Status Bar）显示的通知文本提示，例如：

```
notification.tickerText = "hello";
```

- 发出提示音，例如：

```
notification.defaults |= Notification.DEFAULT_SOUND;
notification.sound = Uri.parse("file:///sdcard/notification/ringer.mp3");
notification.sound = Uri.withAppendedPath(Audio.Media.INTERNAL_CONTENT_URI, "6");
```

- 手机振动，例如：

```
notification.defaults |= Notification.DEFAULT_VIBRATE;
long[] vibrate = {0,100,200,300};
notification.vibrate = vibrate;
```

- LED 灯闪烁，例如：

```
notification.defaults |= Notification.DEFAULT_LIGHTS;
notification.ledARGB = 0xff00ff00;
notification.ledOnMS = 300;
notification.ledOffMS = 1000;
notification.flags |= Notification.FLAG_SHOW_LIGHTS;
```

4）发送通知，代码如下。

```
private static final int ID_NOTIFICATION = 1;
mNotificationManager.notify(ID_NOTIFICATION, notification);
```

2．更新通知

如果需要更新一个通知，只需要在设置好 Notification 之后，再调用 setLatestEventInfo，然后重新发送一次通知即可。



为了更新一个已经触发过的 Notification，传入相同的 ID。用户既可以传入相同的 Notification 对象，也可以是一个全新的对象。只要 ID 相同，新的 Notification 对象会替换状态条图标和扩展的状态窗口的细节。

另外，还可以使用 ID 来取消 Notification，通过调用 NotificationManager 的 cancel 方法，代码如下。

```
notificationManager.cancel(notificationRef);
```

当取消一个 Notification 时，会移除它的状态条图标以及清除在扩展的状态窗口中的信息。

6.8.3　联合使用Toast和Notification

在本节的内容中，将通过一个具体实例的实现过程，来讲解联合使用 Toast 和 Notification 实现提醒功能效果的具体使用流程。本实例源代码保存在“光盘:\daima\6\toast_and_notification”，具体实现流程如下。

第 1 步：打开 eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“toast_and_notification”的工程文件。

第 2 步：编写 main.xml 主文件，此文件是一个布局文件，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="介绍 Notification" />
    <Button android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="介绍 Toast" />
</LinearLayout>
```

通过上述代码插入了两个 Button 按钮，执行后效果如图 6-63 所示。

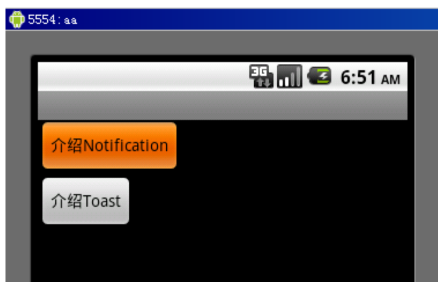


图 6-63　插入两个 Button

第 3 步：编写处理文件 ActivityMain.java，具体代码如下所示。

```
package com.eoeandroid.toast_and_notification;
```

```
import com.eoeandroid.toast_and_notification.R;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
public class ActivityMain extends Activity {
        OnClickListener listener1 = null;
        OnClickListener listener2 = null;

        Button button1;
        Button button2;
        @Override
        public void onCreate(Bundle savedInstanceState) {
                super.onCreate(savedInstanceState);
                listener1 = new OnClickListener() {
                        public void onClick(View v) {
                                setTitle("介绍 Notification");
                                Intent intent = new Intent(ActivityMain.this,
                                                ActivityMainNotification.class);
                                startActivity(intent);
                        }
                };
                listener2 = new OnClickListener() {
                        public void onClick(View v) {
                                setTitle("介绍 Toast");
                                Intent intent = new Intent(ActivityMain.this,
                                                ActivityToast.class);
                                startActivity(intent);
                        }
                };
                setContentView(R.layout.main);
                button1 = (Button) findViewById(R.id.button1);
                button1.setOnClickListener(listener1);
                button2 = (Button) findViewById(R.id.button2);
                button2.setOnClickListener(listener2);
        }
}
```

在上述代码中，对两个 Button 绑定了单击监听器 OnClickListener，当单击这两个 Button 时，会跳转到新的 Activity 上面。

第 4 步：编写第一个 Button 的处理程序，即单击图 6-64 中的“介绍 Notification”按钮后，执行 ActivityMainNotification.java，其主要代码如下所示。

```
package com.eoeandroid.toast_and_notification;
```



```
import com.eoeandroid.toast_and_notification.R;
import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
public class ActivityMainNotification extends Activity {
    private static int NOTIFICATIONS_ID = R.layout.activity_notification;
    private NotificationManager mNotificationManager;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_notification);
        Button button;
        mNotificationManager = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);

        button = (Button) findViewById(R.id.sun_1);
        button.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                setWeather("适合户外", "天气状况", "适合户外", R.drawable.sun);
            }
        });

        button = (Button) findViewById(R.id.cloudy_1);
        button.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                setWeather("不太适合", "天气状况", "不太适合", R.drawable.cloudy);
            }
        });

        button = (Button) findViewById(R.id.rain_1);
        button.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                setWeather("不适合", "天气状况", "不适合", R.drawable.rain);
            }
        });

        button = (Button) findViewById(R.id.defaultSound);
        button.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                setDefault(Notification.DEFAULT_SOUND);
            }
        });
```

```
        button = (Button) findViewById(R.id.defaultVibrate);
        button.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                setDefault(Notification.DEFAULT_VIBRATE);
            }
        });

        button = (Button) findViewById(R.id.defaultAll);
        button.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                setDefault(Notification.DEFAULT_ALL);
            }
        });

        button = (Button) findViewById(R.id.clear);
        button.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                mNotificationManager.cancel(NOTIFICATIONS_ID);
            }
        });

    }

    private void setWeather(String tickerText, String title, String content,
            int drawable) {

        Notification notification = new Notification(drawable, tickerText,
                System.currentTimeMillis());

        PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
                new Intent(this, ActivityMain.class), 0);

        notification.setLatestEventInfo(this, title, content, contentIntent);

        mNotificationManager.notify(NOTIFICATIONS_ID, notification);
    }

    private void setDefault(int defaults) {

        PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
                new Intent(this, ActivityMain.class), 0);

        String title = "天气预报";
        String content = "晴空万里";
```



```
            final Notification notification = new Notification(R.drawable.sun,
                    content, System.currentTimeMillis());

            notification.setLatestEventInfo(this, title, content, contentIntent);

            notification.defaults = defaults;

            mNotificationManager.notify(NOTIFICATIONS_ID, notification);
        }
    }
```

下面开始对上述代码进行讲解。

1）因为所有的 Notification 都是通过 NotificationManager 来管理的，所以应该首先创建 NotificationManager 对象实例，以便管理这个 Activity 中信息。

```
mNotificationManager=(NotificationManager)getSystemService(NOTIFICATION_SERVICE);
```

2）函数 setWeather 是 ActivityMainNotification 中的重要函数之一，它实例化了一个 Notification，并将这个 Notification 显示出来。

3）看下面的代码。

```
Notification notification = new Notification(drawable, tickerText,
                System.currentTimeMillis());
```

包含了 3 个参数，具体说明如下。

- 第一个：要显示的图片的 ID。
- 第二个：显示的文本文字。
- 第三个：Notification 显示的时间，一般是立即显示，时间就是 System.currentTimeMillis()。

4）函数 setDefault 也是 ActivityMainNotification.java 中的一个重要函数，在此函数中初始化了一个 Notification，在设置 Notification 时使用了其默认值的形式，即：

```
notification.defaults = defaults;
```

另外，在上述程序中还用到了以下几种表现形式。

- Notification.DEFAULT_VIBRATE：表示当前的 Notification 显示出来时手机会发出震动。
- Notification.DEFAULT_SOUND：表示当前的 Notification 显示出来时手机会伴随音乐。
- Notification.DEFAULT_ALL：表示当前的 Notification 显示出来时手机即会震动，也会伴随音乐。

这样单击第一个 Button 后会执行上述处理程序，来到对应的新界面，新界面的布局文件是由 activity_notification.xml 实现的，其主要代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
```

```
android:layout_height="fill_parent">

<LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content">

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">

        <Button
            android:id="@+id/sun_1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="适合户外活动" />

        <Button
            android:id="@+id/cloudy_1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="不太适合户外活动" />

          <Button
            android:id="@+id/rain_1"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="一点也不适合户外活动" />

    </LinearLayout>

      <TextView
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_marginTop="20dip"
        android:text="高级 notification" />

    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">

        <Button
            android:id="@+id/defaultSound"
            android:layout_width="wrap_content"
```



```
                android:layout_height="wrap_content"
                android:text="有声音的 notification" />

            <Button
                android:id="@+id/defaultVibrate"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="振动的 notification" />

             <Button
                android:id="@+id/defaultAll"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="声音+振动的 notification" />

        </LinearLayout>

        <Button android:id="@+id/clear"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:layout_marginTop="20dip"
            android:text="清除 notification" />

    </LinearLayout>

</ScrollView>
```

执行后新界面的效果如图 6-64 所示。

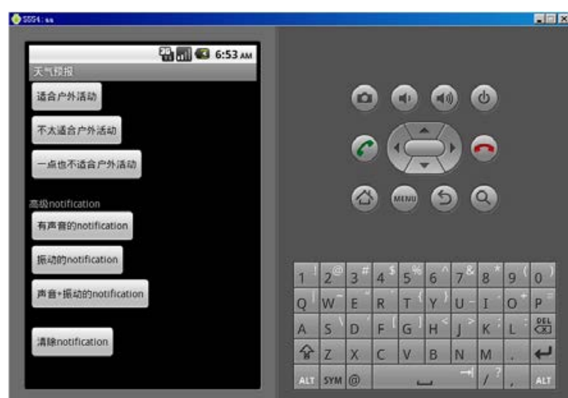


图 6-64　运行效果

当单击新图 6-64 中的 Button 后，会根据上述处理文件实现某种效果，例如，单击“有声音的 Notification”按钮后，会发出声音。

第 5 步：编写第二个 Button 的处理程序，即单击图 6-63 中的“介绍 Toast”按钮后，执行 ActivityToast.java，其主要代码如下所示。

```
package com.eoeandroid.toast_and_notification;

import com.eoeandroid.toast_and_notification.R;

import android.app.Activity;
import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.content.Context;
import android.content.Intent;
import android.os.Bundle;
import android.view.LayoutInflater;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.TextView;
import android.widget.Toast;

public class ActivityToast extends Activity {

    OnClickListener listener1 = null;
    OnClickListener listener2 = null;
    Button button1;
    Button button2;
    private static int NOTIFICATIONS_ID = R.layout.activity_toast;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        listener1 = new OnClickListener() {
            public void onClick(View v) {
                setTitle("短时显示 Toast");
                showToast(Toast.LENGTH_SHORT);

            }
        };
        listener2 = new OnClickListener() {
            public void onClick(View v) {
                setTitle("长时显示 Toast");
                showToast(Toast.LENGTH_LONG);
                showNotification();
            }
        };
        setContentView(R.layout.activity_toast);
        button1 = (Button) findViewById(R.id.button1);
        button1.setOnClickListener(listener1);
```



```
        button2 = (Button) findViewById(R.id.button2);
        button2.setOnClickListener(listener2);
    }

    protected void showToast(int type) {

        View view = inflateView(R.layout.toast);

        TextView tv = (TextView) view.findViewById(R.id.content);
        tv.setText("欢迎加入爬山摄影户外俱乐部，强身健体、增强友谊，共创和谐社会！");
        /*实例化 Toast*/
        Toast toast = new Toast(this);
        toast.setView(view);
        toast.setDuration(type);
        toast.show();
    }

    private View inflateView(int resource) {
        LayoutInflater vi = (LayoutInflater) getSystemService(Context.LAYOUT_INFLATER_SERVICE);
        return vi.inflate(resource, null);
    }

    protected void showNotification() {

        NotificationManager notificationManager = (NotificationManager) getSystemService
(NOTIFICATION_SERVICE);

        CharSequence title = "最专业的户外拓展俱乐部";
        CharSequence contents = "户外合作论坛.com";

        PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
                new Intent(this, ActivityMain.class), 0);

        Notification notification = new Notification(R.drawable.default_icon,
                title, System.currentTimeMillis());

        notification.setLatestEventInfo(this, title, contents, contentIntent);

        // 100ms 延迟后，振动 250ms，停止 100ms 后振动 500ms
        notification.vibrate = new long[] { 100, 250, 100, 500 };

        notificationManager.notify(NOTIFICATIONS_ID, notification);
    }
}
```

上述处理程序是通过 Toast 实现的，它不需要用 NotificationManager 来管理，上述代码的处理流程如下。

1）实例化 Toast，每个 Toast 和一个 View 相关。

2）设置 Toast 的长短，通过“showToast(Toast.LENGTH_SHORT);”来设置 Toast 短时间显示，通过“showToast(Toast.LENGTH_LONG);”来设置 Toast 长时间显示。

3）通过函数 showToast 来显示 Toast，具体说明如下。

- 当单击“长时显示”按钮后，程序会长时间地显示提醒，并用 Notification 在状态栏中提示用户。
- 当单击“短时显示”按钮后，程序会短时间的显示提醒。

这样单击第二个 Button 后会执行上述处理程序，来到对应的新界面，新界面的布局文件是由 activity_toast.xml 实现的，其主要代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <Button android:id="@+id/button1"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="短时显示 Toast" />
    <Button android:id="@+id/button2"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="长时显示 Toast" />
</LinearLayout>
```

执行后新界面的效果如图 6-65 所示。

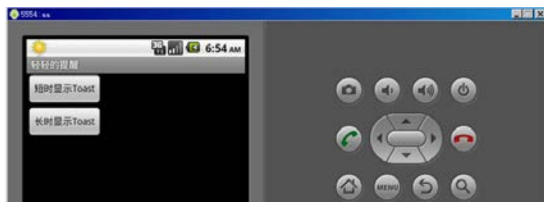


图 6-65　运行效果

当单击新图 6-65 中的 Button 后，会根据上述处理文件而实现某种效果，例如，单击“短时显示 Toast”按钮后，会短时间显示一个提醒，当单击“长时显示 Toast”按钮后，会长时间显示一个提醒，两种方式的提醒界面都是相同的，具体效果如图 6-66 所示。

至此，整个实例介绍完毕，读者可以阅读随书光盘中的源代码，来了解更加具体的信息。



图 6-66　运行效果

第 7 章　Android组件高级应用

在第六章中，详细讲解了 Android 中主要组件的基本知识。实际上，除了前面介绍的组件之外，还有很多其他组件，并且利用这些组件能够实现更为复杂的功能。在本章的内容中，将进一步讲解 Android 组件的高级应用知识，为学习本书后面知识打下基础。

7.1　对话框中的进度条

对话框中的进度条（ProgressDialog）的功能是在对话框中实现显示进度条效果。在本节的内容中，将通过一个具体实例的实现过程，来讲解 ProgressDialog 的基本使用方法。本实例保存在“光盘\daima\7\jindu”中，具体实现流程如下：

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“jindu”的工程文件。

第 2 步：编写布局文件 main.xml，用于在界面中插入两个 Button 按钮，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
  <Button
    android:id="@+id/Button01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="圆形进度条" />
  <Button
    android:id="@+id/Button02"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="长形进度条" />
</LinearLayout>
```

第 3 步：编写文件 strings.xml，用于设置标题文本，主要代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">对话框进度条演示</string>
    <string name="app_name">对话框进度条</string>
</resources>
```

第 4 步：编写文件 Activity01.java，用于实现单击 Button 按钮后的事件处理程序，具体实现流程如下。

1）声明进度条对话框 m_pDialog。

2）得到按钮对象 mButton01 和 mButton02。

3）设置 mButton01 的事件监听，首先创建 ProgressDialog 对象，然后设置进度条风格，风格为圆形、旋转的，最后分别设置 ProgressDialog 标题、提示信息、图标等信息。

4）设置 mButton02 的事件监听，首先创建 ProgressDialog 对象，然后设置进度条风格，风格为长形，最后分别设置 ProgressDialog 标题、提示信息、图标等信息。

5）让 ProgressDialog 显示出来。

文件 Activity01.java 的具体实现代码如下所示。

```
package com.yarin.android.jindu;

import android.app.Activity;
import android.app.ProgressDialog;
import android.content.DialogInterface;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;

public class Activity01 extends Activity
{
    private Button mButton01,mButton02;

    int m_count = 0;
    //声明进度条对话框
    ProgressDialog m_pDialog;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //得到按钮对象
        mButton01 = (Button)findViewById(R.id.Button01);
        mButton02 = (Button)findViewById(R.id.Button02);
```



```
    //设置 mButton01 的事件监听
    mButton01.setOnClickListener(new Button.OnClickListener() {
        @Override
        public void onClick(View v)
        {
            // TODO Auto-generated method stub

            //创建 ProgressDialog 对象
            m_pDialog = new ProgressDialog(Activity01.this);

            // 设置进度条风格，风格为圆形，旋转的
            m_pDialog.setProgressStyle(ProgressDialog.STYLE_SPINNER);

            // 设置 ProgressDialog 标题
            m_pDialog.setTitle("提示");

            // 设置 ProgressDialog 提示信息
            m_pDialog.setMessage("这是一个圆形进度条对话框");

            // 设置 ProgressDialog 标题图标
            m_pDialog.setIcon(R.drawable.img1);

            // 设置 ProgressDialog 的进度条是否不明确
            m_pDialog.setIndeterminate(false);

            // 设置 ProgressDialog 是否可以按退回按键取消
            m_pDialog.setCancelable(true);

            // 设置 ProgressDialog 的一个 Button
            m_pDialog.setButton("确定", new DialogInterface.OnClickListener() {
                public void onClick(DialogInterface dialog, int i)
                {
                    //点击“确定”按钮取消对话框
                    dialog.cancel();
                }
            });

            // 让 ProgressDialog 显示
            m_pDialog.show();
        }
    });

//设置 mButton02 的事件监听
    mButton02.setOnClickListener(new Button.OnClickListener() {
        @Override
```

```
public void onClick(View v)
{
    // TODO Auto-generated method stub

    m_count = 0;

    // 创建 ProgressDialog 对象
    m_pDialog = new ProgressDialog(Activity01.this);

    // 设置进度条风格，风格为长形
    m_pDialog.setProgressStyle(ProgressDialog.STYLE_HORIZONTAL);

    // 设置 ProgressDialog 标题
    m_pDialog.setTitle("提示");

    // 设置 ProgressDialog 提示信息
    m_pDialog.setMessage("这是一个长形对话框进度条");

    // 设置 ProgressDialog 标题图标
    m_pDialog.setIcon(R.drawable.img2);

    // 设置 ProgressDialog 进度条进度
    m_pDialog.setProgress(100);

    // 设置 ProgressDialog 的进度条是否不明确
    m_pDialog.setIndeterminate(false);

    // 设置 ProgressDialog 是否可以按退回按键取消
    m_pDialog.setCancelable(true);

    // 让 ProgressDialog 显示
    m_pDialog.show();

    new Thread()
    {
        public void run()
        {
            try
            {
                while (m_count <= 100)
                {
                    // 由线程来控制进度。
                    m_pDialog.setProgress(m_count++);
                    Thread.sleep(100);
                }
                m_pDialog.cancel();
```



```
                                    }
                                    catch (InterruptedException e)
                                    {
                                            m_pDialog.cancel();
                                    }
                            }
                    }.start();

            }
        });
    }
}
```

至此，整个实例介绍完毕，执行后的效果如图 7-1 所示。



图 7-1　执行效果

当单击图 7-1 中的“图形进度条”按钮后，将显示图形化的进度条效果，如图 7-2 所示；当单击图 7-1 中的“长形进度条”按钮后，将显示长形化的进度条效果，如图 7-3 所示。



图 7-2　图形化进度条



图 7-3　长形化进度条

7.2　再看布局组件

在上一章的内容中，已经简单讲解了 Android 的布局知识。因为布局在 Android 中是一个最基本处理功能，所以本节将再次介绍一下，使读者加深对相关知识的理解。



7.2.1 Android的五大布局对象

Android 中有五大布局对象，分别是 FrameLayout(框架布局)、LinearLayout(线性布局)、AbsoluteLayout(绝对布局)、RelativeLayout(相对布局)、TableLayout(表格布局)。

1．FrameLayout

FrameLayout 是最简单的一个布局对象，它被定制为用户屏幕上的一个空白备用区域，之后可以在其中填充一个单一对象，例如，一张要发布的图片。所有的子元素将会固定在屏幕的左上角；不能为 FrameLayout 中的一个子元素指定一个位置。后一个子元素将会直接在前一个子元素上进行覆盖填充，把它们部分或全部挡住（除非后一个子元素是透明的）。看下面的 main.xml 文件代码。

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >

  <!-- 我们在这里加了一个 Button 按钮 -->
<Button
      android:text="button"
      android:layout_width="fill_parent"
      android:layout_height="wrap_content"
/>
<TextView
      android:text="textview"
      android:textColor="#0000ff"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
/>
</FrameLayout>
```

上述代码就是一个用 FrameLayout 布局实现的，执行后效果如图 7-4 所示。

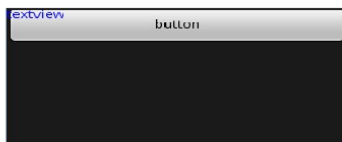


图 7-4 FrameLayout 布局

2．LinearLayout

LinearLayout 能够根据用户为它设置的垂直或水平的属性值，来排列所有的子元素。所有的子元素都被堆放在其他元素之后，因此一个垂直列表的每一行只会有一个元素，而不管他们有多宽，而一个水平列表将会只有一个行高(高度为最高子元素的高度加上边框高度)。LinearLayout 保持子元素之间的间隔以及互相对齐（相对一个元素的右对齐、中间对齐或者左对齐）。

LinearLayout 还支持为单独的子元素指定 weight（权值），好处是允许子元素可以填充屏



幕上的剩余空间。这样也避免了在一个大屏幕中，一串小对象挤成一堆的情况，而是允许它们放大填充空白。子元素指定一个 Weight 值，剩余的空间就会按这些子元素指定的 Weight 比例分配给这些子元素。默认的 Weight 值为 0。例如，如果有 3 个文本框，其中两个指定了 Weight 值为 1，那么，这两个文本框将等比例地放大，并填满剩余的空间，而第三个文本框不会放大。

看下面的 main.xml 文件代码。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
   <LinearLayout
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="2">
    <TextView
        android:text="Welcome to Mr Wei's blog"
        android:textSize="15pt"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
     />
    </LinearLayout>
    <LinearLayout
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1">

    <TextView
        android:text="red"

        android:gravity="center_horizontal" //这里字水平居中
        android:background="#aa0000"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_weight="1"/>
    <TextView
        android:text="green"
        android:gravity="center_horizontal "
        android:background="#00aa00"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_weight="1"/>
    </LinearLayout>
```




```
</LinearLayout>
```

上述代码就是一个用 LinearLayout 布局实现的，执行后效果如图 7-5 所示。

3．AbsoluteLayout

AbsoluteLayout 可以让子元素指定准确的 x/y 坐标值，并显示在屏幕上。(0, 0)为左上角，当向下或向右移动时，坐标值将变大。AbsoluteLayout 没有页边框，允许元素之间互相重叠（尽管不推荐）。在日常项目应用中，通常不推荐使用 AbsoluteLayout，除非有正当理由要使用它，因为它会使界面代码太过刚性，以至于在不同的设备上可能不能很好地工作。看下面的 main.xml 文件代码。

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<EditText
    android:text="Welcome to Mr Wei's blog"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
/>
<Button
    android:layout_x="250px"                //设置按钮的 X 坐标
    android:layout_y="40px"                 //设置按钮的 Y 坐标
    android:layout_width="70px"         //设置按钮的宽度
    android:layout_height="wrap_content"
    android:text="Button"
/>
</AbsoluteLayout>
```

上述代码就是一个用 AbsoluteLayout 布局实现的，执行后效果如图 7-6 所示。



图 7-5　LinearLayout 布局

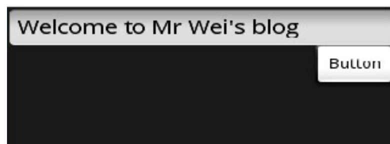


图 7-6　AbsoluteLayout 布局

4．RelativeLayout

RelativeLayout 允许子元素指定它们相对于其他元素或父元素的位置（通过 ID 指定）。为此可以以右对齐，或上下，或置于屏幕中央的形式来排列两个元素。元素按顺序排列，因此如果第一个元素在屏幕的中央，那么相对于这个元素的其他元素将以屏幕中央的相对位置来排列。如果使用 XML 来指定这个 layout，在用户定义它之前，被关联的元素必须定义。看



下面的 main.xml 文件代码。

```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="Welcome to Mr Wei's blog:"/>
    <EditText
        android:id="@+id/entry"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_below="@id/label"/>
    <Button
        android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dip"
        android:text="OK" />
    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/ok"
        android:layout_alignTop="@id/ok"
        android:text="Cancel" />
</RelativeLayout>
```

上述代码就是一个用 RelativeLayout 布局实现的，执行后效果如图 7-7 所示。



图 7-7　RelativeLayout 布局

5．TableLayout

TableLayout 将子元素的位置分配到行或列中。一个 TableLayout 由许多的 TableRow 组成，每个 TableRow 都会定义一个 row（事实上，用户可以定义其他的子对象，这在下面会解释到）。TableLayout 容器不会显示 Row（行）、Cloumns（列）或 Cell（单元格）的边框线。每个 Row 拥有 0 个或多个的 Cell ；每个 Cell 拥有一个 View 对象。表格由列和行组成许多的单元格。表格允许单元格为空。单元格不能跨列，这与 HTML 中的不一样。看下面的 main.xml 文件代码。

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent" android:layout_height="fill_parent"
    android:stretchColumns="1">
    <TableRow>
        <TextView android:layout_column="1" android:text="Open..." />
        <TextView android:text="Ctrl-O" android:gravity="right" />
    </TableRow>
    <TableRow>
        <TextView android:layout_column="1" android:text="Save..." />
        <TextView android:text="Ctrl-S" android:gravity="right" />
    </TableRow>
    //这里是上图中的分隔线
    <View android:layout_height="2dip" android:background="#FF909090" />
    <TableRow>
        <TextView android:text="X" />
        <TextView android:text="Export..." />
        <TextView android:text="Ctrl-E" android:gravity="right " />
    </TableRow>
    <View android:layout_height="2dip" android:background="#FF909090" />
    <TableRow>
        <TextView android:layout_column="1" android:text="Quit"
            android:padding="3dip" />
    </TableRow>
</TableLayout>
```

上述代码就是一个用 TableLayout 布局实现的，执行后效果如图 7-8 所示。



图 7-8　TableLayout 布局

7.2.2　垂直线性布局

在本节的内容中，将通过一个实例的实现过程，来讲解垂直线性布局（vertical LinearLayout）的基本使用方法。本实例保存在“光盘\daima\7\chuizhixian”中，具体实现流程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“chuizhixian”的工程文件。

第 2 步：编写布局文件 main.xml，在里面插入 4 个 TextView 控件，分别显示“首行”，“次行”，“三行”，“四行”，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
```



```
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        >
         <TextView
             android:text="首行"
             android:gravity="center_vertical"
             android:textSize="15pt"
             android:background="#aa0000"
             android:layout_width="fill_parent"
             android:layout_height="wrap_content"
             android:layout_weight="1"/>

         <TextView
             android:text="次行"
             android:textSize="15pt"
             android:gravity="center_vertical"
             android:background="#00aa00"
             android:layout_width="fill_parent"
             android:layout_height="wrap_content"
             android:layout_weight="1"/>

         <TextView
             android:text="三行"
             android:textSize="15pt"
             android:gravity="center_vertical"
             android:background="#0000aa"
             android:layout_width="fill_parent"
             android:layout_height="wrap_content"
             android:layout_weight="1"/>
         <TextView
             android:text="四行"
             android:textSize="15pt"
             android:gravity="center_vertical"
             android:background="#aaaa00"
             android:layout_width="fill_parent"
             android:layout_height="wrap_content"
             android:layout_weight="1"/>

</LinearLayout>
```

第 3 步：编写文件 strings.xml，用于设置标题文本，主要代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">雨夜你好</string>
    <string name="app_name">欢迎你</string>
```

```
    </resources>
```

第 4 步：编写文件 Activity01.java，用于实现界面布局。此文件的具体实现比较简单，其主要实现代码如下所示。

```
package com.yarin.android.chuizhixian;

import android.app.Activity;
import android.os.Bundle;

public class Activity01 extends Activity
{
	/** Called when the activity is first created. */
	@Override
	public void onCreate(Bundle savedInstanceState)
	{
		super.onCreate(savedInstanceState);
		setContentView(R.layout.main);
	}
}
```

至此，整个实例介绍完毕，执行后的效果如图 7-9 所示。

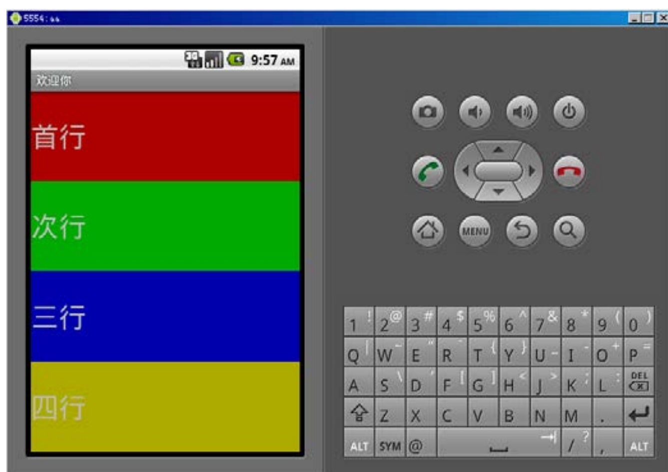


图 7-9　执行效果

7.2.3　水平线性布局

在本节的内容中，将通过一个实例的实现过程，来讲解水平线性布局（horizontal LinearLayout）的基本使用方法。本实例保存在“光盘:\daima\7\shuipingxian”中，具体实现流程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“shuipingxian”的工程文件。



第 2 步：编写布局文件 main.xml，在里面插入 4 个 TextView 控件，分别显示“第一列”、“第二列”、“第三列”、“第四列”，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
     <TextView
         android:text="第一列"
         android:gravity="center_horizontal"
         android:background="#aa0000"
         android:layout_width="wrap_content"
         android:layout_height="fill_parent"
         android:layout_weight="1"/>

     <TextView
         android:text="第二列"
         android:gravity="center_horizontal"
         android:background="#00aa00"
         android:layout_width="wrap_content"
         android:layout_height="fill_parent"
         android:layout_weight="1"/>

     <TextView
         android:text="第三列"
         android:gravity="center_horizontal"
         android:background="#0000aa"
         android:layout_width="wrap_content"
         android:layout_height="fill_parent"
         android:layout_weight="1"/>

     <TextView
         android:text="第四列"
         android:gravity="center_horizontal"
         android:background="#aaaa00"
         android:layout_width="wrap_content"
         android:layout_height="fill_parent"
         android:layout_weight="1"/>
</LinearLayout>
```

第 3 步：编写文件 strings.xml，用于设置标题文本，主要代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">你好</string>
```

```
        <string name="app_name">E 雨夜！ </string>
</resources>
```

第 4 步：编写文件 Activity01.java，用于实现界面布局。此文件的具体实现比较简单，其主要实现代码如下所示。

```
package com.yarin.android.shuipingxian;

import android.app.Activity;
import android.os.Bundle;

public class Activity01 extends Activity
{
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState)
        {
                super.onCreate(savedInstanceState);
                setContentView(R.layout.main);
        }
}
```

至此，整个实例介绍完毕，执行后的效果如图 7-10 所示。

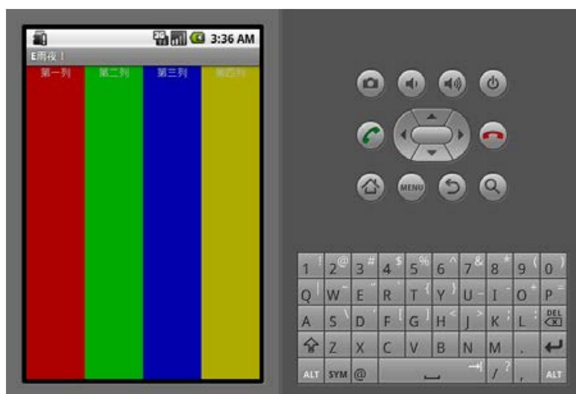


图 7-10　执行效果

7.2.4　相对布局

在本节的内容中，将通过一个实例的实现过程，来讲解相对布局（RelativeLayout）的基本使用方法。本实例保存在“光盘\daima\7\xiangduibu”中，具体实现流程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“xiangduibu”的工程文件。

第 2 步：编写布局文件 main.xml，在里面分别插入 1 个 TextView 控件，1 个 EditText 控件，两个 Button 控件,具体代码如下所示。



```
<?xml version="1.0" encoding="utf-8"?>
<RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">

    <TextView
        android:id="@+id/label"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="可以输入:"/>

    <EditText
        android:id="@+id/entry"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:background="@android:drawable/editbox_background"
        android:layout_below="@id/label"/>

    <Button
        android:id="@+id/ok"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_below="@id/entry"
        android:layout_alignParentRight="true"
        android:layout_marginLeft="10dip"
        android:text="确定" />

    <Button
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:layout_toLeftOf="@id/ok"
        android:layout_alignTop="@id/ok"
        android:text="取消" />

</RelativeLayout>
```

第 3 步：编写文件 strings.xml，用于设置标题文本，主要代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">你好</string>
    <string name="app_name">雨夜</string>
</resources>
```

第 4 步：编写文件 Activity01.java，用于实现界面布局。此文件的具体实现比较简单，其主要实现代码如下所示。

```
package com.yarin.android.xiangduibu;

import android.app.Activity;
import android.os.Bundle;

public class Activity01 extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

至此，整个实例介绍完毕，执行后的效果如图 7-11 所示。



图 7-11　执行效果

7.2.5　表单布局

在本节的内容中，将通过一个实例的实现过程，来讲解表单布局（TableLayout）的基本使用方法。本实例保存在“光盘\daima\7\biaodan”中，具体实现流程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“biaodan”的工程文件。

第 2 步：编写布局文件 main.xml，在里面插入 7 个 TextView 控件，6 个 TableLayout 控件，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<TableLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:stretchColumns="1">
    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="打开..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-O"
            android:gravity="right"
```



```
            android:padding="3dip" />
    </TableRow>
    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="保存..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-S"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>
    <TableRow>
        <TextView
            android:layout_column="1"
            android:text="另存为..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-Shift-S"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>

    <View
        android:layout_height="2dip"
        android:background="#FF909090" />
    <TableRow>
        <TextView
            android:text="*"
            android:padding="3dip" />
        <TextView
            android:text="导入..."
            android:padding="3dip" />
    </TableRow>
    <TableRow>
        <TextView
            android:text="*"
            android:padding="3dip" />
        <TextView
            android:text="导出..."
            android:padding="3dip" />
        <TextView
            android:text="Ctrl-E"
            android:gravity="right"
            android:padding="3dip" />
    </TableRow>
```

```
        <View
            android:layout_height="2dip"
            android:background="#FF909090" />

        <TableRow>
            <TextView
                android:layout_column="1"
                android:text="退出"
                android:padding="3dip" />
        </TableRow>
    </TableLayout>
```

第 3 步：编写文件 strings.xml，用于设置标题文本，主要代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">你好</string>
    <string name="app_name">雨夜</string>
</resources>
```

第 4 步：编写文件 Activity01.java，用于实现界面布局。此文件的具体实现比较简单，其主要实现代码如下所示。

```
package com.yarin.android.biaodan;

import android.app.Activity;
import android.os.Bundle;

public class Activity01 extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
    }
}
```

至此，整个实例介绍完毕，执行后的效果如图 7-12 所示。



图 7-12　执行效果



7.2.6　切换卡

在本节的内容中，将通过一个实例的实现过程，来讲解切换卡（TabWidget）的基本使用方法。本实例保存在“光盘\daima\7\qiehuan”中，具体实现流程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“qiehuan”的工程文件。

第 2 步：编写布局文件 main.xml，在里面分别插入 3 个 TextView 控件，1 个 TabWidget 控件，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<TabHost xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@android:id/tabhost"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <TabWidget
            android:id="@android:id/tabs"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" />
        <FrameLayout
            android:id="@android:id/tabcontent"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent">
            <TextView
                android:id="@+id/textview1"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:text="this is a tab" />
            <TextView
                android:id="@+id/textview2"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:text="this is another tab" />
            <TextView
                android:id="@+id/textview3"
                android:layout_width="fill_parent"
                android:layout_height="fill_parent"
                android:text="this is a third tab" />
     </FrameLayout>
    </LinearLayout>
</TabHost>
```

第 3 步：编写文件 strings.xml，用于设置标题文本，主要代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">你好</string>
    <string name="app_name">雨夜</string>
</resources>
```

第 4 步：编写文件 Activity01.java，用于实现界面布局，具体实现过程如下。

1）声明 TabHost 对象。

2）获取 TabHost 对象。

3）为 TabHost 添加标签，然后新建一个 newTabSpec(newTabSpec)，并设置其标签、图标和内容。

4）分别设置 TabHost 的背景颜色、背景图片资源和当前显示哪一个标签。

5）定义标签切换事件处理方法 setOnTabChangedListener。

文件 Activity01.java 的具体实现代码如下所示。

```
package com.yarin.android.qiehuan;

import android.app.AlertDialog;
import android.app.Dialog;
import android.app.TabActivity;
import android.content.DialogInterface;
import android.graphics.Color;
import android.os.Bundle;
import android.widget.TabHost;
import android.widget.TabHost.OnTabChangeListener;

public class Activity01 extends TabActivity
{
    //声明 TabHost 对象
    TabHost mTabHost;
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        //取得 TabHost 对象
        mTabHost = getTabHost();

        /* 为 TabHost 添加标签 */
        //新建一个 newTabSpec(newTabSpec)
        //设置其标签和图标(setIndicator)
        //设置内容(setContent)
       mTabHost.addTab(mTabHost.newTabSpec("tab_test1")
```



```
                .setIndicator("TAB 1",getResources().getDrawable(R.drawable.img1))
                .setContent(R.id.textview1));
        mTabHost.addTab(mTabHost.newTabSpec("tab_test2")
                .setIndicator("TAB 2",getResources().getDrawable(R.drawable.img2))
                .setContent(R.id.textview2));
        mTabHost.addTab(mTabHost.newTabSpec("tab_test3")
                .setIndicator("TAB 3",getResources().getDrawable(R.drawable.img3))
                .setContent(R.id.textview3));

        //设置 TabHost 的背景颜色
        mTabHost.setBackgroundColor(Color.argb(150, 22, 70, 150));
        //设置 TabHost 的背景图片资源
        //mTabHost.setBackgroundResource(R.drawable.bg0);

        //设置当前显示哪一个标签
        mTabHost.setCurrentTab(0);

        //标签切换事件处理，setOnTabChangedListener
        mTabHost.setOnTabChangedListener(new OnTabChangeListener()
        {
         // TODO Auto-generated method stub
            @Override
            public void onTabChanged(String tabId)
            {
                Dialog dialog = new AlertDialog.Builder(Activity01.this)
                            .setTitle("提示")
                            .setMessage("当前选中："+tabId+"标签")
                            .setPositiveButton("确定",
                            new DialogInterface.OnClickListener()
                            {
                                public void onClick(DialogInterface dialog, int whichButton)
                                {
                                    dialog.cancel();
                                }
                            }).create();//创建按钮

                dialog.show();
            }
        });
    }
}
```

至此，整个实例介绍完毕，执行后的效果如图 7-13 所示。单击某个图片后会运行相应单击事件处理程序，显示对应的对话框。例如，单击第二个图片后，显示的对话框效果如图 7-14 所示。



图 7-13　执行效果



图 7-14　显示对话框

7.3　联合使用Spinner和setDropDownViewResource

在前面的内容中，已经讲解了 Spinner（列表）控件的基本使用方法。在本节的内容中，将通过一个实例的实现过程，来讲解联合使用 Spinner 和 setDropDownViewResource 的基本方法。本实例保存在“光盘:\daima\7\Spinnergaoji”中。

在本实例中，使用了 ArrayAdapter(Context context, int textViewResourceId, T[] objects) 这个 Constructor，textViewResourceId 使用 Android 提供的 ResourceID，objects 为必须传递的字符串数组（String Array）。

Adapter 的 setDropDownViewResource 可以设置下拉菜单的显示方式，将该 xml 定义在 res/layout 目录下面，可针对下拉菜单中的 TextView 进行设置，如同本程序里的 R.layout.myspinner_dropdown 即为自定义的下拉菜单 TextView 样式。除了改变下拉菜单样式外，也对 Spinner 做了一点动态效果，点击 Spinner 时，晃动 Spinner 再出现下拉菜单（myAnimation）。

本实例的具体实现流程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“Spinnergaoji”的工程文件。

第 2 步：编写布局文件 main.xml，用于在界面中分别插入 1 个 Button 控件和 1 个 TextView 控件。具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/white"
    >
    <TextView
    android:id="@+id/myTextView"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/title"
    android:textColor="@drawable/black"
    >
    </TextView>
    <Spinner
```



```
    android:id="@+id/mySpinner"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    >
    </Spinner>
</LinearLayout>
```

通过上述代码，在界面中添加了 UI 元素。

第 3 步：在 res 目录下新建一个 anim 文件夹，存放动画效果用，在其中新建一个 my_anim.xml 文件，具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<set xmlns:android="http://schemas.android.com/apk/res/android">
        <translate
              android:fromXDelta="0"
              android:toXDelta="-100%p"
               android:duration="300">
        </translate>                                    <!--位置转换动画效果  -->
        <alpha
              android:fromAlpha="3.0"
              android:toAlpha="6.0"
               android:duration="300">
        </alpha>                                              <!--透明度转换动画效果  -->
</set>
```

第 4 步：编写 color.xml，设置界面的颜色，具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <drawable name="black">#000000</drawable>
    <drawable name="white">#FFFFFFFF</drawable>
</resources>
```

第 5 步：在 res 目录下的 layout 文件夹中新建一个 myspinner_dropdown.xml 文件，用来存放下拉菜单弹出内容的布局，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:id="@+id/text1"
      android:layout_width="wrap_content"
      android:layout_height="24sp"
      android:singleLine="true"
      style="?android:attr/spinnerDropDownItemStyle"
 />
```

第 6 步：编写事件处理文件 Spinnergaoji.java，其具体实现流程如下。

1）定义一个下拉菜单，以 findViewById()取得对象。

2）定义一个字符串数组和一个 ArrayAdepter，用于显示供选择的国家。

3）给下拉菜单内容设置样式。

4）给下拉菜单设置内容适配器。

5）给下拉菜单添加动画。

文件 Spinnergaoji.java 的具体实现代码如下所示。

```
package irdc.Spinnergaoji;

import irdc.Spinnergaoji.R;
import android.app.Activity;
import android.os.Bundle;
import android.view.MotionEvent;
import android.view.View;
import android.view.animation.Animation;
import android.view.animation.AnimationUtils;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.TextView;
import android.widget.ListView;
import android.widget.Spinner;

public class Spinnergaoji extends Activity
{
   private static final String[] countriesStr =
   { "美国", "英国", "日本", "韩国" };
   private TextView myTextView;
   private Spinner mySpinner;
   private ArrayAdapter<String> adapter;
   Animation myAnimation;

   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      /*  载入 main.xml Layout */
      setContentView(R.layout.main);

      /*  以 findViewById()取得对象  */
      myTextView = (TextView) findViewById(R.id.myTextView);
      mySpinner = (Spinner) findViewById(R.id.mySpinner);

      adapter = new ArrayAdapter<String>(this,
            android.R.layout.simple_spinner_item, countriesStr);
      /* myspinner_dropdown 为自定义下拉菜单模式定义在 res/layout 目录下  */
```



```
adapter.setDropDownViewResource(R.layout.myspinner_dropdown);

/* 将 ArrayAdapter 添加 Spinner 对象中 */
mySpinner.setAdapter(adapter);

/* 将 mySpinner 添加 OnItemSelectedListener */
mySpinner.setOnItemSelectedListener
 (new Spinner.OnItemSelectedListener()
{
   @Override
   public void onItemSelected
     (AdapterView<?> arg0, View arg1, int arg2,
        long arg3)
   {
      /* 将所选 mySpinner 的值代入 myTextView 中 */
      myTextView.setText("您选择的是" + countriesStr[arg2]);
      /* 将 mySpinner 显示 */
      arg0.setVisibility(View.VISIBLE);
   }

   @Override
   public void onNothingSelected(AdapterView<?> arg0)
   {
      // TODO Auto-generated method stub
   }
});

/* 取得 Animation 定义在 res/anim 目录下 */
myAnimation = AnimationUtils.loadAnimation(this, R.anim.my_anim);

/* 将 mySpinner 添加 OnTouchListener */
mySpinner.setOnTouchListener(new Spinner.OnTouchListener()
{

   @Override
   public boolean onTouch(View v, MotionEvent event)
   {
      /* 将 mySpinner 运行 Animation */
      v.startAnimation(myAnimation);
      /* 将 mySpinner 隐藏 */
      v.setVisibility(View.INVISIBLE);
      return false;
   }

});
```

```
            mySpinner.setOnFocusChangeListener(new Spinner.OnFocusChangeListener()
            {
               @Override
               public void onFocusChange(View v, boolean hasFocus)
               {
                  // TODO Auto-generated method stub
               }
            });
        }
    }
```

至此，整个实例介绍完毕。执行后的效果如图 7-15 所示，当单击下拉框时会弹出一个浮动的可选选项框，在此用户可以选择一个国家选项，如图 7-16 所示。

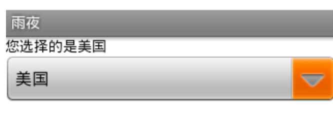


图 7-15　执行效果

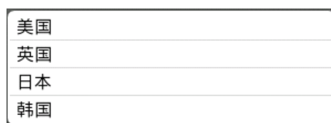


图 7-16　显示浮动框

Animation 主要有两种动态方式，一种是 Tweened Animation（渐变动画），另一种是 Frame By Frame Animation（画面转换动画）。Tweened Animation 则有以下 4 种基本转换方式。

1）AlphaAnimation (transparency changes)：透明度转换。

2）RotateAnimation (rotations)：旋转转换。

3）ScaleAnimation (growing or shrinking)：缩放转换。

4）TranslateAnimation (position changes)：位置转换。

定义好想要的动画 xml 这件后，然后用 AnimationUtils.loadAnimation 将动画加载，在想要加上动态效果的组件中使用 startAnimation 方法。

7.4　Gallery和衍生BaseAdapter容器

在本节的内容中，将通过一个实例的实现过程，来讲解 Gallery（画廊）控件和衍生 BaseAdapter 基本使用方法，本实例保存在“光盘:\daima\7\Galleryjia”中。

在本书第 6 章中已经讲解了 Gallery 的使用方法，为了简化问题，使用了 Android 默认的 Icon 作为 Gallery 显示的内容。现在，将数张 PNG 图片导入 Drawable 当中，在 onCreate 函数响应的载入到 Gallery Widget 中，试着再添加一个 OnItemClick 的事件，以取得图片的 ID 编号来响应用户点击图片时的状态，完成 Gallery 的高级使用。本范例的另一个重点，就是如何设置 Gallery 图片的宽高以及放置图片 Layout 的大小，在此改写一个继承自 BaseAdapter 的 ImageAdapter 容器来存放图片，通过 ImageView.setScaleType() 的方法来改变图片的显示，再通过 setLayoutParams() 方法来改变 Layout 的宽高。

本实例的具体实现流程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“Galleryjia”的工程文件。



第 2 步：编写布局文件 main.xml，添加一个 Gallery 控件和一个 ImageView 控件实现整体布局。具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<Gallery xmlns:android="http://schemas.android.com/apk/res/android"
   android:id="@+id/mygallery"
   android:layout_width="fill_parent"
   android:layout_height="wrap_content"
/>
```

第 3 步：定义 Layout 外部 Resource（资源）的 xml 文件，用来改变 Layout 的背景，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
      <declare-styleable name="Gallery">
            <attr name="android:galleryItemBackground" />
      </declare-styleable>
      <!-- 定义 layout 外部 resource 的 xml 文件，用来改变 layout 的背景图。 -->
</resources>
```

第 4 步：新建一个 myImageAdapter 类——Gallery 的适配器，它继承于 BaseAdapter 类，具体代码如下。

```
public class myImageAdapter extends BaseAdapter {
        @Override
        public int getCount() {
                // TODO Auto-generated method stub
                return 0;
        }
        @Override
        public Object getItem(int position) {
                // TODO Auto-generated method stub
                return null;
        }
        @Override
        public long getItemId(int position) {
                // TODO Auto-generated method stub
                return 0;
        }
        @Override
        public View getView(int position, View convertView, ViewGroup parent) {
                // TODO Auto-generated method stub
                return null;
        }
}
```

第 5 步：修改 mainActivity.java，添加 Gallery 相关操作，主要代码如下所示。

```
public class Galleryjia extends Activity
{
   /** Called when the activity is first created. */
   @Override
   public void onCreate(Bundle savedInstanceState)
   {
      super.onCreate(savedInstanceState);
      setContentView(R.layout.main);
      /*通过 findViewById 取得*/
      Gallery g = (Gallery) findViewById(R.id.mygallery);
      /* 添加一 ImageAdapter 并设置给 Gallery 对象 */
      g.setAdapter(new ImageAdapter(this));

      /* 设置一个 itemclickListener 并 Toast 被点击图片的位置 */
      g.setOnItemClickListener(new OnItemClickListener()
      {
         public void onItemClick
         (AdapterView<?> parent, View v, int position, long id)
         {
            Toast.makeText
            (Galleryjia.this, getString(R.string.my_gallery_text_pre)
            + position+ getString(R.string.my_gallery_text_post),
            Toast.LENGTH_SHORT).show();
         }
      });
   }

   /* 改写 BaseAdapter 自定义一 ImageAdapter class */
   public class ImageAdapter extends BaseAdapter
   {
      /*声明变量*/
      int mGalleryItemBackground;
      private Context mContext;

      /*ImageAdapter 的构造器*/
      public ImageAdapter(Context c)
      {
         mContext = c;

         /* 使用在 res/values/attrs.xml 中的<declare-styleable>定义
         * 的 Gallery 属性.*/
         TypedArray a = obtainStyledAttributes(R.styleable.Gallery);

         /*取得 Gallery 属性的 Index id*/
         mGalleryItemBackground = a.getResourceId
```



```
    (R.styleable.Gallery_android_galleryItemBackground, 0);

    /*让对象的 styleable 属性能够反复使用*/
    a.recycle();
}

/*  覆盖的方法 getCount,返回图片数目  */
public int getCount()
{
    return myImageIds.length;
}

/*  覆盖的方法 getItemId,返回图像的数组 id */

public Object getItem(int position)
{
    return position;
}
public long getItemId(int position)
{
    return position;
}

/*  覆盖的方法 getView,  返回一 View 对象  */
public View getView
(int position, View convertView, ViewGroup parent)
{
    /*产生 ImageView 对象*/
    ImageView i = new ImageView(mContext);
    /*设置图片给 imageView 对象*/
    i.setImageResource(myImageIds[position]);
    /*重新设置图片的宽高*/
    i.setScaleType(ImageView.ScaleType.FIT_XY);
    /*重新设置 Layout 的宽高*/
    i.setLayoutParams(new Gallery.LayoutParams(136, 88));
    /*设置 Gallery 背景图*/
    i.setBackgroundResource(mGalleryItemBackground);
    /*返回 imageView 对象*/
    return i;
}

/*建构一 Integer array 并取得预加载 Drawable 的图片 id*/
private Integer[] myImageIds =
{
    R.drawable.photo1,
    R.drawable.photo2,
```

```
            R.drawable.photo3,
            R.drawable.photo4,
            R.drawable.photo5,
            R.drawable.photo6,
        };
    }
}
```

至此，整个实例介绍完毕。执行后会实现相册效果，如图 7-17 所示；当选择一张图片后，此图片会放大显示，并显示此图片的标号，如图 7-18 所示。

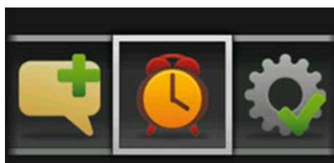


图 7-17　执行效果



图 7-18　显示某张图片

在 Android:ScaleType 中定义了下列常数可供使用，通过“ObjectView.ScaleType 常数名称”的方式，就可以改变图片的显示方式。

- Matrix：0
- fitXY：1
- fitStart：2
- fitCenter：3
- fitEnd：4
- Center：5
- centerCrop：6
- centerInside：7

另外，在主程序中还使用了下面的这段代码。

```
TypedArray a = obtainStyledAttributes(R.styleable.Gallery);
```

这是一个引用自制 Layout 元素的用法，必须在 res/values 目录下添加一个 attrs.xml，并在其中定义 <declare-styleable> 标签 TAG，目的是自定义 Layout 的背景风格，并且通过 TypeArray 的特性，让相同的 Layout 元素可以重复用于每一张图片。

7.5　文件搜索引擎FileSearch

读者应该都用过操作系统中或网页中的文件搜索功能吧，它可以快速帮用户找到想要的文件或资料。如何要在手机中实现文件搜索的功能呢？Java I/O 的 API 中提供了 java.io.File 对象，只要利用 File（文件）对象的方法，再搭配 Android 的 EditText、TextView 等对象，就可以轻松制作出一个手机的文件搜索引擎。



在本节的内容中，将通过一个实例的实现过程，来讲解文件搜索引擎 File Sanch 的基本使用方法。本实例保存在“光盘\daima\7\Galleryjia”中。本范例中使用 EditText、Button 与 TextView3 种对象来实现此功能，用户将要搜索的文件名称或关键字输入 EditText 中，点击 Button 后，程序会在根目录中寻找符合的文件，并将搜索结果显示于 TextView 中；如果找不到符合的文件，则显示找不到文件。

本实例的具体实现流程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“sousuo”的工程文件。

第 2 步：编写布局文件 main.xml，分别添加两个 TextView 控件，1 个 EditText 控件和 1 个 Button 控件，实现整体布局，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
  <AbsoluteLayout
    android:id="@+id/layout1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:background="@drawable/white"
    xmlns:android="http://schemas.android.com/apk/res/android"
  >
    <EditText
      android:id="@+id/mKeyword"
      android:layout_width="198px"
      android:layout_height="wrap_content"
      android:textSize="18sp"
      android:layout_x="110px"
      android:layout_y="12px"
    >
    </EditText>
    <TextView
      android:id="@+id/mText"
      android:layout_width="wrap_content"
      android:layout_height="31px"
      android:textSize="18sp"
      android:layout_x="7px"
      android:layout_y="25px"
      android:text="@string/str_title"
      android:textColor="@drawable/black"
    >
    </TextView>
    <Button
      android:id="@+id/mButton"
      android:layout_width="86px"
      android:layout_height="48px"
      android:text="@string/str_button"
      android:layout_x="100px"
```

```
            android:layout_y="72px"
        >
        </Button>
        <TextView
            android:id="@+id/mResult"
            android:layout_width="296px"
            android:layout_height="283px"
            android:layout_x="10px"
            android:layout_y="132px"
            android:textColor="@drawable/blue"
        >
        </TextView>
    </AbsoluteLayout>
```

第 3 步：在 string.xml 添加程序中要使用的字符串，具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
    <resources>
        <string name="hello">HI</string>
        <string name="app_name">雨夜</string>
        <string name="str_button">请搜索</string>
        <string name="str_title">关键字：</string>
    </resources>
```

第 4 步：编写 sousuo.java，实现搜索功能，其具体实现流程如下。

1）声明对象变量。

2）载入 main.xml Layout。

3）初始化对象，并为 mButton 按钮设置响应单击事件 onClickListener。

4）取得输入的关键字。

5）根据搜索文件的 Method（方法）和关键字搜索。

文件 sousuo.java 的具体实现代码如下所示。

```
package irdc.sousuo;

/* import 相关 class */
import irdc.sousuo.R;

import java.io.File;
import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.TextView;

public class sousuo extends Activity
```



```
{
  /*声明对象变量*/
  private Button mButton;
  private EditText mKeyword;
  private TextView mResult;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    /* 载入 main.xml Layout */
    setContentView(R.layout.main);

    /* 初始化对象 */
    mKeyword=(EditText)findViewById(R.id.mKeyword);
    mButton=(Button)findViewById(R.id.mButton);
    mResult=(TextView) findViewById(R.id.mResult);

    /* 将 mButton 添加 onClickListener */
    mButton.setOnClickListener(new Button.OnClickListener()
    {
      public void onClick(View v)
      {
      /*取得输入的关键字*/
        String keyword = mKeyword.getText().toString();
        if(keyword.equals(""))
        {
          mResult.setText("请勿输入空白的关键字!!");
        }
        else
        {
          mResult.setText(searchFile(keyword));
        }
      }
    });
  }

  /* 搜索文件的 method */
  private String searchFile(String keyword)
  {
    String result="";
    File[] files=new File("/").listFiles();
    for( File f : files )
    {
      if(f.getName().indexOf(keyword)>=0)
```

```
                {
                    result+=f.getPath()+"\n";
                }
            }
            if(result.equals("")) result="找不到文件!!";
            return result;
        }
    }
```

至此，整个实例介绍完毕，执行后的初始效果如图 7-19 所示；当输入关键字并单击“请搜索”按钮后会检索出对应的信息，如图 7-20 所示。



图 7-19　执行效果



图 7-20　搜索的信息

在本实例中，searchFile(String keyword)方法为了搜索根目录下符合关键字的文件，在搜索文件的过程中，只搜索根目录中的文件，并没有再对子目录下的文件作进一步的比较，如果要再强化这个文件搜索的功能，让它也能搜索包含子目录下的所有文件，可以在程序中利用 File.isDirectory() 这个方法来判断其是否为目录。如果是的话，就继续往下一层寻找；不是的话，就终止向下寻找的动作。这个做法在后面的范例中会有详细的示范，要注意的是手机硬件环境是否能负荷程序做大规模的文件搜索，毕竟手机的硬件配备（处理器、内存）是比不上一般计算机的。

另外，在范例中仅针对关键字做了空白检查，在比较文件名称时也没有忽略大小写（要大小写完全符合才能搜索出来），如果要让这个搜索功能更完备，当然可以多做一点变化，例如，检查关键字是否包含特殊的字符、可选择比较文件时是否忽略大小写、可选择要搜索的文件类型，甚至是可自己指定要搜索的目录等，就看各位要如何运用了。

7.6　用AnalogClock和DigitalClock控件实现模拟小时钟

在本节的内容中，将通过一个实例的实现过程，来讲解使用 AnalogClock 和 DigitalClock 实现模拟小时钟的基本流程。本实例保存在“光盘:\daima\7\shizhong”中。

Android 里的 AnalogClock Widget 是一个时钟对象，本范例将配置一个小时钟，并在其下放置一个 TextView，为了做对照，上面放置的为模拟时钟，下面的 TextView 则模拟电子时钟，将 AnalogClock 的时间以数字钟形式显示。

本实例的难点是：android.os.Handler、java.lang.Thread 以及 android.os.Message 3 个对象的整合应用，通过产生 Thread 对象，在进程内同步调用 System.currentTimeMillis()来获取系统时间，并通过 Message 对象来通知 Handler 对象；Handler 则扮演联系 Activity 与 Thread 之间



的桥梁，在收到 Message 对象后，将时间变量的值显示于 TextView 当中，产生数字时钟的外观与功能。

本实例的具体实现流程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建 1 个名为“shizhong”的工程文件。

第 2 步：编写布局文件 main.xml，分别添加 1 个 AnalogClock 控件、1 个 DigitalClock 控件、1 个 TextView 控件，实现整体布局，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<AbsoluteLayout
    android:id="@+id/widget0"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android">
<TextView
    android:id="@+id/TextView_showTime"
    android:layout_width="200px"
    android:layout_height="39px"
    android:textSize="25px"
    android:layout_x="82px"
    android:layout_y="222px">
</TextView>
<AnalogClock
    android:id="@+id/Clock"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_x="82px"
    android:layout_y="34px">
</AnalogClock>
<DigitalClock
    android:id="@+id/DigitalClock01"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="上午 1:01"
    android:textSize="25px"
    android:layout_x="82px"
    android:layout_y="300px">
</DigitalClock>
<TextView
    android:id="@+id/widget46"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:text="模拟时钟"
    android:layout_x="11px"
    android:layout_y="46px">
```

```
    </TextView>
    <TextView
        android:id="@+id/widget47"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="线程时钟"
        android:layout_x="11px"
        android:layout_y="228px">
    </TextView>
    <TextView
        android:id="@+id/widget48"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="数字时钟"
        android:layout_x="11px"
        android:layout_y="308px">
    </TextView>
    </AbsoluteLayout>
```

第 3 步：模拟时钟的实现不需要额外代码，只需要在 UI 中添加，其自动显示时间。具体代码如下。

```
/*定义模拟时钟对象*/
AnalogClock myClock;
/*从 XML 中获取模拟时钟 UI 对象*/
myClock=(AnalogClock)findViewById(R.id.Clock);
```

第 4 步：数字时钟的实现也不需要额外代码，只需要在 UI 中添加，其自动显示时间。具体代码如下。

```
/*定义数字时钟对象*/
DigitalClock myDigClock;
/*从 XML 中获取数字时钟 UI 对象*/
myDigClock=(DigitalClock)findViewById(R.id.DigitalClock01);
```

使用线程实现的 TextView 时钟则需要线程 Thread、Handler（发送、处理消息）辅助实现，具体代码如下所示。

```
import android.os.Handler;
import android.os.Message;
public class Clock extends Activity implements Runnable{
    public Handler myHandler;
    @Override
    protected void onCreate(Bundle savedInstanceState) {
        // TODO Auto-generated method stub
        super.onCreate(savedInstanceState);
        setContentView(R.layout.gh);
```



```
            myHandler = new Handler() {
                @Override
                public void handleMessage(Message msg) {
                    // TODO Auto-generated method stub
                }
            };
            Thread myT = new Thread(this);
            myT.start();
        }
        @Override
        public void run() {
            // TODO Auto-generated method stub
        }
    }
```

第 5 步：实现最终的主程序 shizhong.java，因为要另外加载 Java 的 Calendar 与 Thread 对象，所以需要在 onCreate() 中构造 Handler 与 Thread 两对象，并实现 handelMessage() 与 run() 两个方法。具体代码如下所示。

```
package irdc.shizhong;

import android.app.Activity;
import android.os.Bundle;
/*这里我们需要使用 Handler 类与 Message 类来处理运行线程*/
import android.os.Handler;
import android.os.Message;
import android.widget.AnalogClock;
import android.widget.TextView;
/*需要使用 Java 的 Calendar 与 Thread 类来取得系统时间*/
import irdc.shizhong.R;

import java.util.Calendar;
import java.lang.Thread;

public class shizhong extends Activity
{
  /*声明一常数作为判别信息用*/
  protected static final int GUINOTIFIER = 0x1234;

  /*声明两个 widget 对象变量*/
  private TextView mTextView;
  public AnalogClock mAnalogClock;

  /*声明与时间相关的变量*/
  public Calendar mCalendar;
  public int mMinutes;
```




```
public int mHour;

/*声明关键 Handler 与 Thread 变量*/
public Handler mHandler;
private Thread mClockThread;

/** Called when the activity is first created. */
public void onCreate(Bundle savedInstanceState)
{
   super.onCreate(savedInstanceState);
   setContentView(R.layout.main);

   /*通过 findViewById 取得两个 widget 对象*/
   mTextView=(TextView)findViewById(R.id.myTextView);
   mAnalogClock=(AnalogClock)findViewById(R.id.myAnalogClock);

   /*通过 Handler 来接收运行线程所传递的信息并更新 TextView*/
   mHandler = new Handler()
   {
      public void handleMessage(Message msg)
      {
         /*这里是处理信息的方法*/
         switch (msg.what)
         {
            case shizhong.GUINOTIFIER:
            /*  在这要处理 TextView 对象 Show 时间的事件  */
               mTextView.setText(mHour+" : "+mMinutes);
               break;
         }
         super.handleMessage(msg);
      }
   };

   /*通过运行线程来持续取得系统时间*/
    mClockThread=new LooperThread();
    mClockThread.start();
}

/*改写一个 Thread Class 用来持续取得系统时间*/
class LooperThread extends Thread
{
   public void run()
   {
      super.run();
      try
      {
```



```
                do
                {
                   /*取得系统时间*/
                   long time = System.currentTimeMillis();
                   /*通过 Calendar 对象来取得小时与分钟*/
                   final Calendar mCalendar = Calendar.getInstance();
                   mCalendar.setTimeInMillis(time);
                   mHour = mCalendar.get(Calendar.HOUR);
                   mMinutes = mCalendar.get(Calendar.MINUTE);

                   /*让运行线程休息一秒*/
                   Thread.sleep(1000);
                    /*关键程序:取得时间后发出信息给 Handler*/
                    Message m = new Message();
                    m.what = shizhong.GUINOTIFIER;
                    shizhong.this.mHandler.sendMessage(m);
                }while(shizhong.LooperThread.interrupted()==false);
                /*当系统发出中断信息时停止本循环*/
             }
             catch(Exception e)
             {
                e.printStackTrace();
             }
          }
       }
    }
```

至此，整个实例介绍完毕。执行后会显示一个时钟效果，如图 7-21 所示。

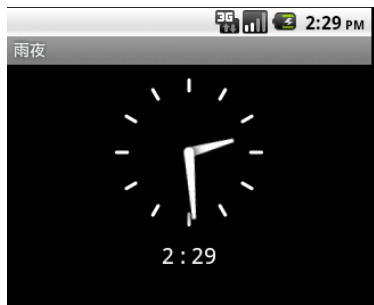


图 7-21　运行效果

在本实例代码中，实际上要达到本范例效果的代码应该只有两行，也就是将对于 TextView 与进程 Thread 的处理，改为使用 widget.DigitalClock 的方式，具体写法如下。

```
import android.widget.AnalogClock
mDigitalClock =(DigitalClock)findViewById(R.id.myDigitalClock);
```

而本程序使用 TextView 来模拟 DigitalClock 的做法，实际上，也是参考 AnalogClock 与

DigitalClock 这两个 Widget（构件）的程序代码所做的练习，对于读者在后面实现 Timer 相关的小对象，会有所帮助。

Android 提供了 System.currentTimeMillis()、uptimeMillis()、elapsedRealtime()这 3 种不同特性的系统时钟给开发者使用。本范例使用 System.currentTimeMillis() 就是标准的 Clock 用法，需要搭配真实的日期与时间使用，另外两者则是用于 interval 与 elapse time 来控制程序与 UI，读者可以自行练习查看效果。

7.7 使用ListActivity（活动列表）

ListActivity 也是一个独立的类，它和 Activity 是同一级别的类。ListActivity 也能够实现布局处理，能用于显示菜单列表、列表明细项目。本节将详细介绍 ListActivity 的基本知识。

7.7.1 ListActivity介绍

在 android.app 包里包含了几个重要的类，ListActivity 就是其中之一。ListActivity 类其实就是一个含有一个 ListView 组件的 Activity 类。也就是说，如果直接在一个普通的 Activity 中加一个 ListView，也是完全可以取代 ListActivity 的，只是 ListActivity 更方便而已。

一个像数组或者是光标一样，通过绑定数据资源来陈列一系列选项的活动。当选择这些选项时，会触发一个事件。ListActivity 主持操作着一个列表视图对象，这个列表视图能绑定不同的数据资源，典型的就是一个持有查询结果的数组或者是光标。

1．屏幕布局(Screen Layout)

ListActivity 有一个默认的布局，这个布局由单一的、全屏列表构成。通过在 onCreate() 中使用 setContentView()设置视图布局来定制自己的屏幕布局。如果要完成这些，用户自己的视图必须包含一个 ID 为“@android:id/list”的 ListView 对象。如果用户的列表为空时，可以包含另外一个视图对象，这个空的列表必须有一个“android:empty”值的 ID，注意到当有一个空的视图显示时，这个列表视图将会在没有任何数据时被隐藏。

2．排布局(Row Layout)

可以在列表中确定单一的排，当是通过在活动所操作的 ListAdapter 对象中设定的一个资源布局来实现的。一个 ListAdapter 持有一个参数来为每一行确定所使用到的需要布局的资源。同时，它还有另外两个参数，这两个参数让用户明确与哪个对象相互关联，通常是两个平行的数组。Android 提供了一些标准的行布局资源。这些都是在 R.layout 类中所定义的，名称像 simple_list_item_1,simple_list_item_2,two_line_list_item 一样。

下面先来分析一段代码，这段代码就是使用了这个方面的知识。

```
String songss[];
    //首先是创建一个 File 类型的实例对象 home
    File home = new File(MEDIA_PATH);
    //列出目录文件中的所有文件并将这些文件名称放到字符串数组中
    songss=home.list();
    //下面是关键代码，创建一个 ArrayAdapter 对象，并将 3 个参数分别置为 this(表明是当前上下文),
//R.layout.song_item 表明要在 song_item 中显示),songss(这就是显示在 song_item 中的内容)，这样
就将要显示在 ListView 中的内容设置好了。
```



```
ArrayAdapter<String> songList = new ArrayAdapter<String>(this,R.layout.song_item,songss);
//使用 setListAdapter()，就是使 ArrayAdapter 类型的 songList 运行，即真正起作用。
setListAdapter(songList);
```

通过上述代码介绍了 ListView 中 ListAdapter 的使用。

同时，Android 不仅仅提供了数组，还提供了另外两个标准的列表适配器：处理 static 数据类型的 SimpleAdapter 和处理光标结果查询的 SimpleCursorAdapter。例如，下面是一个将名称和公司信息绑定到一个两行排布局的活动列表视图（所谓的两行排指的是两行数值作为一个排布局，也可以是三行作为一个排布局或者是一行作为一个行布局）。

```
public class MyListAdapter extends ListActivity {
    @Override
    protected void onCreate(Bundle icicle){
        super.onCreate(icicle);
        setContentView(R.layout.custom_list_activity_view);
        mCursor = People.query(this.getContentResolver(), null);
        startManagingCursor(mCursor);
        ListAdapter adapter = new SimpleCursorAdapter(
                this,
                android.R.layout.two_line_list_item,
                mCursor,
                new String[] {People.NAME, People.COMPANY},
                new int[]);
        setListAdapter(adapter);
    }
}
```

3．可实现的方法

如果让程序继承于 ListActivity，则可以使用下面的方法。

- getListAdapter()：获取列表项目的 Adapter。
- getListView：获取列表的 View。
- getSelectItemId()：获取当前 Keypad 所选择的 Item ID。
- onContentChanged()：ListActivity 列表内容更新事件。
- onListItemClick(ListView,View,int,long)：User 在列表项目中单击触发事件。
- onRestoreInstancsState(Bundle)：还原至此实例状态事件。
- setListAdapter(ListAdapter)：设置 ListAdapter 的列表项目。
- setSelection(int)：设置所选的项目。

在使用 ListActivity 时，并不用像使用 Activity 那样必须使用 setContentView 来设置版型 Layout 才能显示页面。ListActivity 不需要重写 protected void onCreate(Bundle savedInstanceState)的情况下，直接将列表加载到 ListActivity 中，这样使用起来十分方便，经常用于投票选择和多项目列表条。

7.7.2 ListActivity使用实例

在本节的内容中，将通过一个实例的实现过程，来讲解使用 ListActivity 的具体流程。本实例保存在“光盘\daima\7\shiyongListActivity”中，是通过 ListActivity 和 Menu 联合实现的。具体实现流程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“shiyongListActivity”的工程文件。

第 2 步：编写布局文件 main.xml 实现整体布局，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<TextView
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/myTextView1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
/>
```

第 3 步：在 string.xml 添加程序中要使用的字符串，具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">你好雨夜</string>
    <string name="app_name">欢迎你</string>
    <string name="str_list1">精绝古城</string>
    <string name="str_list2">龙岭迷窟</string>
    <string name="str_list3">云南虫谷  </string>
    <string name="str_list4">昆仑神宫</string>
    <string name="str_list5">C++语言</string>
    <string name="str_list6">Java 语言</string>
    <string name="str_list7">PHP 语言</string>
    <string name="str_list8">Basic 语言</string>
    <string name="str_menu_list1">显示列表 1</string>
    <string name="str_menu_list2">显示列表 2</string>
</resources>
```

第 4 步：编写 shiyongListActivity.java，实现显示功能，其具体实现流程如下。

1）声明对象变量。

2）载入 main.xml Layout。

3）通过 ordor MenuIteml 设置数组内的元素顺序。

4）通过 get Resourcesc,方法获取数组资源。

5）用 on Options Item Selected 显示用户所选数组选项。

文件 shiyongListActivity.java 的具体实现代码如下所示。

```
package irdc.shiyongListActivity;
```



```
import irdc.shiyongListActivity.R;
import android.app.ListActivity;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.ListView;
import android.widget.Toast;

public class shiyongListActivity extends ListActivity
{
    private int selectedItem = -1;
    private String[] mString;
    static final private int MENU_LIST1 = Menu.FIRST;
    static final private int MENU_LIST2 = Menu.FIRST+1;
    private ArrayAdapter<String> mla;

    @Override
    protected void onCreate(Bundle savedInstanceState)
    {
        // TODO 自动引起的方法残余部分
        super.onCreate(savedInstanceState);
    }

    @Override
    protected void onListItemClick(ListView l, View v, int position, long id)
    {
        // TODO 自动引起的方法残余部分

        selectedItem = position;
        Toast.makeText(shiyongListActivity.this, mString[selectedItem], Toast.LENGTH_SHORT).show();
        super.onListItemClick(l, v, position, id);
    }

    @Override
    public boolean onCreateOptionsMenu(Menu menu)
    {
        // TODO Auto-generated method stub
        /* menu 组 ID */
        int idGroup1 = 0;

        /*项目的顺序位置*/
        int orderMenuItem1 = Menu.NONE;
        int orderMenuItem2 = Menu.NONE+1;
```

```
            menu.add(idGroup1, MENU_LIST1, orderMenuItem1, R.string.str_menu_list1);
            menu.add(idGroup1, MENU_LIST2, orderMenuItem2, R.string.str_menu_list2);

            return super.onCreateOptionsMenu(menu);
        }

        @Override
        public boolean onOptionsItemSelected(MenuItem item)
        {
            // TODO Auto-generated method stub
            switch(item.getItemId())
            {
                case (MENU_LIST1):
                    mString = new String[]
                                            {
                                                getResources().getString(R.string.str_list1),
                                                getResources().getString(R.string.str_list2),
                                                getResources().getString(R.string.str_list3),
                                                getResources().getString(R.string.str_list4)
                                            };

                    mla = new ArrayAdapter<String>(shiyongListActivity.this, R.layout.main, mString);
                    shiyongListActivity.this.setListAdapter(mla);
                    break;
                case (MENU_LIST2):
                    mString = new String[]
                                            {
                                                getResources().getString(R.string.str_list5),
                                                getResources().getString(R.string.str_list6),
                                                getResources().getString(R.string.str_list7),
                                                getResources().getString(R.string.str_list8)
                                            };
                    mla = new ArrayAdapter<String>(shiyongListActivity.this, R.layout.main, mString);
                    shiyongListActivity.this.setListAdapter(mla);
                    break;
            }
            return super.onOptionsItemSelected(item);
        }

    }
```

至此，整个实例介绍完毕。执行后将在底部显示两个布局块，并分别显示设置的文本，具体效果如图 7-22 所示；当单击第一个布局块后会显示弹出对应的信息，如图 7-23 所示；当单击第二个布局块后会显示弹出对应的信息，如图 7-24 所示。



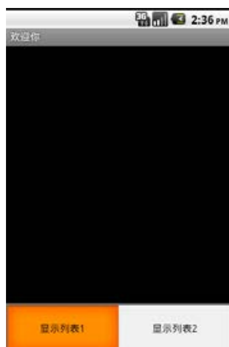


图 7-22　初始效果



图 7-23　列表 1 的信息

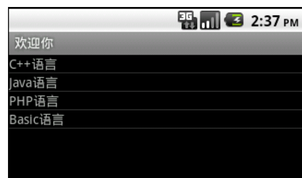


图 7-24　列表 2 的信息

7.8　绘图处理

在 Android 平台中，有一个专门用于绘图处理的类：android.Graphics 底层图形类，它用于实现绘图处理功能。在本节的内容中，将简要讲解绘图类 Graphics 的基本知识，并详细讲解 Matrix 实现图形变换的基本知识。

7.8.1　graphics类基础

1．android.graphics.Matrix

Matrix 能够实现图形的变换操作，如常见的缩放和旋转处理。Matrix 中有关图形的变换、缩放等相关操作常用的方法有如下几种。

1）void reset()：重置一个 Matrix 对象。

2）void set(Matrix src：复制一个源矩阵，和本类的构造方法 Matrix(Matrix src)一样。

3）boolean isIdentity()：返回这个矩阵是否定义(已经有意义)。

4）void setRotate(float degrees)：指定一个角度以（0,0）为坐标进行旋转。

5）void setRotate(float degrees, float px, float py)：指定一个角度以（px,py）为坐标进行旋转。

6）void setScale(float sx, float sy)：缩放处理。

7）void setScale(float sx, float sy, float px, float py)：以坐标（px,py）进行缩放。

8）void setTranslate(float dx, float dy)：平移。

9）void setSkew (float kx, float ky, float px, float py：以坐标（px，py）进行倾斜。

10）void setSkew (float kx, float ky)：倾斜处理。

2．android.graphics.Bitmap

Bitmap 是一个位图操作类，实现对位图的基本操作。Bitmap 中提供了很多实用的方法，其中最为常用的几种方法总结如下。

1）boolean compress(Bitmap.CompressFormat format, int quality, OutputStream stream)：压缩一个 Bitmap 对象根据相关的编码、画质保存到一个 OutputStream 中。其中第一个压缩格式目前有 JPG 和 PNG。

2）void copyPixelsFromBuffer(Buffer src)：从一个 Buffer 缓冲区复制位图像素。

3）void copyPixelsToBuffer(Buffer dst)：将当前位图像素内容复制到一个 Buffer 缓冲区。

4）下述方法用于创建一个可以缩放的位图对象。

```
static Bitmap createBitmap(Bitmap src)
static Bitmap createBitmap(int[] colors, int width, int height, Bitmap.Config config)
static Bitmap createBitmap(int[] colors, int offset, int stride, int width, int height, Bitmap.Config config)
static Bitmap createBitmap(Bitmap source, int x, int y, int width, int height, Matrix m, boolean filter)
static Bitmap createBitmap(int width, int height, Bitmap.Config config)
static Bitmap createBitmap(Bitmap source, int x, int y, int width, int height)
static Bitmap createScaledBitmap(Bitmap src, int dstWidth, int dstHeight, boolean filter)
```

5）final int getHeight()：获取高度。

6）final int getWidth()：获取宽度。

7）final boolean hasAlpha()：是否有透明通道。

8）void setPixel(int x, int y, int color)：设置某像素的颜色。

9）int getPixel(int x, int y)：获取某像素的颜色，Android 开发网提示这里返回的 int 型是 color 的定义。

3．android.graphics.BitmapFactory

作为 Bitmap 对象的 I/O 类，BitmapFactory 类提供了丰富的构造 Bitmap 对象的方法，比如从一个字节数组、文件系统、资源 ID、以及输入流中来创建一个 Bitmap 对象，下面本类的全部成员，除了 decodeFileDescriptor 外其他的重载方法都很常用。

1）从字节数组创建。

```
static Bitmap decodeByteArray(byte[] data, int offset, int length)
static Bitmap decodeByteArray(byte[] data, int offset, int length, BitmapFactory.Options opts)
```

2）从文件创建，路径要写全。

```
static Bitmap decodeFile(String pathName, BitmapFactory.Options opts)
static Bitmap decodeFile(String pathName)
```

3）从输入流句柄创建。

```
static Bitmap decodeFileDescriptor(FileDescriptor fd, Rect outPadding, BitmapFactory.Options opts)
static Bitmap decodeFileDescriptor(FileDescriptor fd)
```

4）从 Android 的 APK 文件资源中创建，Android123 提示是从/res/的 drawable 中。

```
static Bitmap decodeResource(Resources res, int id)
static Bitmap decodeResource(Resources res, int id, BitmapFactory.Options opts)
static Bitmap decodeResourceStream(Resources res, TypedValue value, InputStream is, Rect pad,
BitmapFactory.Options opts)
```

5）从一个输入流中创建。

```
static Bitmap decodeStream(InputStream is)
static Bitmap decodeStream(InputStream is, Rect outPadding, BitmapFactory.Options opts)
```



4．android.graphics.Color

有关 Android 平台上表示颜色的方法有很多种，Color 提供了常规主要颜色的定义比如 Color.BLACK 和 Color.GREEN 等，平时创建时主要使用以下静态方法。

1）static int argb(int alpha, int red, int green, int blue)：构造一个包含透明对象的颜色。

2）static int rgb(int red, int green, int blue)：构造一个标准的颜色对象。

3）static int parseColor(String colorString)：解析一种颜色字符串的值，比如传入 Color.BLACK。

本类返回的均为一个类似于：绿色为 0xff00ff00，红色为 0xffff0000 的值。可以将这个 DWORD 型看做 AARRGGBB，AA 代表 Aphla 透明色，后面的就不难理解，每个分成 WORD 正好为 0～255。

5．android.graphics.NinePatch

NinePatch 是 Android 平台特有的一种非矢量图形自然拉伸处理方法，可以帮助常规的图形在拉伸时不会缩放，实例中 Android 开发网提示大家对于 Toast 的显示就是该原理，同时 SDK 中提供了一个工具名为 Draw 9-Patch，有关该工具的使用方法可以参考《Draw 9-Patch 使用方法介绍》一文。由于该类提供了高质量支持透明的缩放方式，所以图形格式为 PNG，文件命名方式为.9.png 的后缀，比如 android123.9.png。

6．android.graphics.Paint

Paint 类可以理解为画笔、画刷的属性定义，本类中常用的方法如下。

1）void reset()：重置。

2）void setARGB(int a, int r, int g, int b)或 void setColor(int color)：均为设置 Paint 对象的颜色。

3）void setAntiAlias(boolean aa)：是否抗锯齿，需要配合 void setFlags (Paint.ANTI_ALIAS_FLAG) 来帮助消除锯齿使其边缘更平滑。

4）Shader setShader(Shader shader)：设置阴影，Shader 类是一个矩阵对象，如果为 NULL 将清除阴影。

5）void setStyle(Paint.Style style)：设置样式，一般为 FILL 填充，或者 STROKE 凹陷效果。

6）void setTextSize(float textSize)：设置字体大小。

7）void setTextAlign(Paint.Align align)：文本对齐方式。

8）Typeface setTypeface(Typeface typeface)：设置字体，通过 Typeface 可以加载 Android 内部的字体，对于中文一般为宋体，部分 ROM 可以自己添加，如雅黑等。

9）void setUnderlineText(boolean underlineText)：是否设置下划线，需要配合 void setFlags (Paint.UNDERLINE_TEXT_FLAG)方法。

7．android.graphics.Rect

Rect 可以理解为矩形区域，类似的还有 Point 一个点，Rect 类除了表示一个矩形区域位置描述外，Android123 提示主要可以帮助用户计算图形之间是否碰撞(包含)关系，对于 Android 游戏开发比较有用，其主要的成员 Aontains 包含了如下 3 种重载方法，来判断包含关系。

```
boolean contains(int left, int top, int right, int bottom)
boolean contains(int x, int y)
boolean contains(Rect r)
```

8．android.graphics.Region

Region 在 Android 平台中表示一个区域和 Rect 不同的是，它表示的是一个不规则的样子，可以是椭圆、多边形等，而 Rect 仅仅是矩形。同样 Region 的 boolean contains(int x, int y)成员可以判断一个点是否在该区域内。

9．android.graphics.Typeface

Typeface 类是帮助描述一个字体对象，在 TextView 中通过使用 setTypeface 方法来制定一个输出文本的字体，其直接构造调用成员 create 方法可以直接指定一个字体名称和样式，例如：

```
static Typeface create(Typeface family, int style)
static Typeface create(String familyName, int style)
```

同时使用 isBold 和 isItalic 方法可以判断出是否包含粗体或斜体的字型。

```
final boolean   isBold()
final boolean   isItalic()
```

该类的创建方法还有从 apk 的资源或从一个具体的文件路径，其具体方法如下。

```
static Typeface   createFromAsset(AssetManager mgr, String path)
static Typeface   createFromFile(File path)
static Typeface   createFromFile(String path)
```

7.8.2 使用Matrix实现图片缩放

在本节的内容中，将通过一个实例的实现过程，来讲解使用 Matrix 实现图片缩放的具体流程。本实例保存在“光盘\daima\7\suofang”中，其具体实现流程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“suofang”的工程文件。

第 2 步：编写布局文件 main.xml 实现整体布局，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
  <AbsoluteLayout
    android:id="@+id/layout1"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    xmlns:android="http://schemas.android.com/apk/res/android"
  >
    <ImageView
      android:id="@+id/myImageView"
      android:layout_width="200px"
      android:layout_height="150px"
```



```
        android:src="@drawable/suofang"
        android:layout_x="0px"
        android:layout_y="0px"
    >
    </ImageView>
    <Button
        android:id="@+id/myButton1"
        android:layout_width="90px"
        android:layout_height="60px"
        android:text="@string/str_button1"
        android:textSize="18sp"
        android:layout_x="20px"
        android:layout_y="372px"
    >
    </Button>
    <Button
        android:id="@+id/myButton2"
        android:layout_width="90px"
        android:layout_height="60px"
        android:text="@string/str_button2"
        android:textSize="18sp"
        android:layout_x="210px"
        android:layout_y="372px"
    >
    </Button>
</AbsoluteLayout>
```

通过上述代码，在页面中插入了 1 幅图片 suofang.bng，两个 Button 按钮。

第 3 步：在 string.xml 添加程序中要使用的字符串，具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
    <resources>
        <string name="hello">Hello</string>
        <string name="app_name">雨夜</string>
        <string name="str_button1">缩小处理</string>
        <string name="str_button2">放大处理</string>
    </resources>
```

第 4 步：编写处理文件 suofang.java，其具体实现流程如下。

1）载入布局文件 main.xml Layout。

2）取得屏幕分辨率大小，初始化相关变量。

3）实现缩小按钮的单击处理事件 mButton01.setOnClickListener。

4）实现放大按钮的单击处理事件 mButton02.setOnClickListener。

5）定义图片缩小方法 small()。

6）定义图片放大方法 big()。

文件 suofang.java 的具体实现代码如下所示。

```
package irdc.suofang;

/* import 相关 class */
import irdc.suofang.R;
import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Matrix;
import android.os.Bundle;
import android.util.DisplayMetrics;
import android.view.View;
import android.widget.AbsoluteLayout;
import android.widget.Button;
import android.widget.ImageView;

public class suofang extends Activity
{
  /* 相关变量声明 */
  private ImageView mImageView;
  private Button mButton01;
  private Button mButton02;
  private AbsoluteLayout layout1;
  private Bitmap bmp;
  private int id=0;
  private int displayWidth;
  private int displayHeight;
  private float scaleWidth=1;
  private float scaleHeight=1;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    /* 载入 main.xml Layout */
    setContentView(R.layout.main);

    /* 取得屏幕分辨率大小 */
    DisplayMetrics dm=new DisplayMetrics();
    getWindowManager().getDefaultDisplay().getMetrics(dm);
    displayWidth=dm.widthPixels;
    /* 屏幕高度须扣除下方 Button 高度 */
    displayHeight=dm.heightPixels-80;
    /* 初始化相关变量 */
    bmp=BitmapFactory.decodeResource(getResources(),
```



```
                                                    R.drawable.suofang);
        mImageView = (ImageView)findViewById(R.id.myImageView);
        layout1 = (AbsoluteLayout)findViewById(R.id.layout1);
        mButton01 = (Button)findViewById(R.id.myButton1);
        mButton02 = (Button)findViewById(R.id.myButton2);

        /*  缩小按钮 onClickListener */
        mButton01.setOnClickListener(new Button.OnClickListener()
        {
          @Override
          public void onClick(View v)
          {
            small();
          }
        });

        /*  放大按钮 onClickListener */
        mButton02.setOnClickListener(new Button.OnClickListener()
        {
          @Override
          public void onClick(View v)
          {
            big();
          }
        });
      }

      /*  图片缩小的 method */
      private void small()
      {
        int bmpWidth=bmp.getWidth();
        int bmpHeight=bmp.getHeight();
        /*  设置图片缩小的比例  */
        double scale=0.8;
        /*  计算出这次要缩小的比例  */
        scaleWidth=(float) (scaleWidth*scale);
        scaleHeight=(float) (scaleHeight*scale);

        /*  产生 reSize 后的 Bitmap 对象  */
        Matrix matrix = new Matrix();
        matrix.postScale(scaleWidth, scaleHeight);
        Bitmap resizeBmp = Bitmap.createBitmap(bmp,0,0,bmpWidth,
                                                      bmpHeight,matrix,true);

        if(id==0)
        {
```




```
        /* 如果是第一次按，就删除原来默认的 ImageView */
        layout1.removeView(mImageView);
      }
      else
      {
        /* 如果不是第一次按，就删除上次放大缩小所产生的 ImageView */
        layout1.removeView((ImageView)findViewById(id));
      }
      /* 产生新的 ImageView，放入 reSize 的 Bitmap 对象，再放入 Layout 中 */
      id++;
      ImageView imageView = new ImageView(suofang.this);
      imageView.setId(id);
      imageView.setImageBitmap(resizeBmp);
      layout1.addView(imageView);
      setContentView(layout1);

      /* 因为图片放到最大时放大按钮会 disable，所以在缩小时把它重设为 enable */
      mButton02.setEnabled(true);
    }

    /* 图片放大的 method */
    private void big()
    {
      int bmpWidth=bmp.getWidth();
      int bmpHeight=bmp.getHeight();
      /* 设置图片放大的比例 */
      double scale=1.25;
      /* 计算这次要放大的比例 */
      scaleWidth=(float)(scaleWidth*scale);
      scaleHeight=(float)(scaleHeight*scale);

      /* 产生 reSize 后的 Bitmap 对象 */
      Matrix matrix = new Matrix();
      matrix.postScale(scaleWidth, scaleHeight);
      Bitmap resizeBmp = Bitmap.createBitmap(bmp,0,0,bmpWidth,
                                             bmpHeight,matrix,true);

      if(id==0)
      {
        /* 如果是第一次按，就删除原来设置的 ImageView */
        layout1.removeView(mImageView);
      }
      else
      {
        /* 如果不是第一次按，就删除上次放大缩小所产生的 ImageView */
        layout1.removeView((ImageView)findViewById(id));
```



```
            }
            /* 产生新的 ImageView，放入 reSize 的 Bitmap 对象，再放入 Layout 中 */
            id++;
            ImageView imageView = new ImageView(suofang.this);
            imageView.setId(id);
            imageView.setImageBitmap(resizeBmp);
            layout1.addView(imageView);
            setContentView(layout1);

            /* 如果再放大会超过屏幕大小，就把 Button disable */
            if(scaleWidth*scale*bmpWidth>displayWidth||
                scaleHeight*scale*bmpHeight>displayHeight)
            {
              mButton02.setEnabled(false);
            }
          }
        }
```

至此，整个实例介绍完毕。执行后将显示 1 幅图片和 2 个按钮，如图 7-25 所示；分别单击“缩小处理”和“放大处理”按钮后，会实现对图片的缩小、放大处理，如图 7-26 所示。



图 7-25　初始效果



图 7-26　放大后效果

7.8.3　使用Bitmap和Matrix实现图片旋转

在本节的内容中，将通过一个实例的实现过程，来讲解使用 Bitmap 和 Matrix 实现图片旋转的具体流程。本实例保存在“光盘\daima\7\xuanzhuan”中，具体实现流程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“xuanzhuan”的工程文件。

第 2 步：编写布局文件 main.xml 实现整体布局，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
  xmlns:android="http://schemas.android.com/apk/res/android"
  android:background="@drawable/white"
  android:orientation="vertical"
  android:layout_width="fill_parent"
```

```
        android:layout_height="fill_parent"
        >
        <TextView
            android:id="@+id/myTextView1"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/app_name"/>
        <LinearLayout
            android:orientation="horizontal"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
        >
            <Button
                android:id="@+id/myButton1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/str_button1" />
            <ImageView
                android:id="@+id/myImageView1"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_gravity="center" />
            <Button
                android:id="@+id/myButton2"
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:text="@string/str_button2" />
        </LinearLayout>
    </LinearLayout>
```

通过上述代码，在页面中插入了 1 幅图片 hippo.png，两个 Button 按钮，1 个 TextView 文本框。

第 3 步：在 string.xml 添加程序中要使用的字符串，具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="hello">Hello</string>
    <string name="app_name">雨夜</string>
    <string name="str_button1">左旋转</string>
    <string name="str_button2">右旋转</string>
    <string name="str_done">完成</string>
</resources>
```

第 4 步：编写处理文件 xuanzhuan.java，其具体实现流程如下。

1）加载默认的 Drawable。

2）实现向左旋转按钮的处理事件 mButton1.setOnClickListener。



3）实现向右旋转按钮的处理事件 mButton2.setOnClickListener。

文件 xuanzhuan.java 的具体实现代码如下所示。

```
package irdc.xuanzhuan;

import irdc.xuanzhuan.R;
import android.app.Activity;
import android.graphics.Bitmap;
import android.graphics.BitmapFactory;
import android.graphics.Matrix;
import android.graphics.drawable.BitmapDrawable;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.ImageView;
import android.widget.TextView;

public class xuanzhuan extends Activity
{
  private Button mButton1;
  private Button mButton2;
  private TextView mTextView1;
  private ImageView mImageView1;
  private int ScaleTimes;
  private int ScaleAngle;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    mButton1 =(Button) findViewById(R.id.myButton1);
    mButton2 =(Button) findViewById(R.id.myButton2);
    mTextView1 = (TextView) findViewById(R.id.myTextView1);
    mImageView1 = (ImageView) findViewById(R.id.myImageView1);
    ScaleTimes = 1;
    ScaleAngle = 1;

    final Bitmap mySourceBmp =
    BitmapFactory.decodeResource(getResources(), R.drawable.hippo);

    final int widthOrig = mySourceBmp.getWidth();
    final int heightOrig = mySourceBmp.getHeight();
```

```
/* 程序刚运行，加载默认的 Drawable */
mImageView1.setImageBitmap(mySourceBmp);

/* 向左旋转按钮 */
mButton1.setOnClickListener(new Button.OnClickListener()
{
   @Override
   public void onClick(View v)
   {
      // TODO Auto-generated method stub
      ScaleAngle--;
      if(ScaleAngle<-5)
      {
         ScaleAngle = -5;
      }

      /* ScaleTimes=1，维持 1:1 的宽高比例*/
      int newWidth = widthOrig * ScaleTimes;
      int newHeight = heightOrig * ScaleTimes;

      float scaleWidth = ((float) newWidth) / widthOrig;
      float scaleHeight = ((float) newHeight) / heightOrig;

      Matrix matrix = new Matrix();
      /* 使用 Matrix.postScale 设置维度 */
      matrix.postScale(scaleWidth, scaleHeight);

      /* 使用 Matrix.postRotate 方法旋转 Bitmap*/
      //matrix.postRotate(5*ScaleAngle);
      matrix.setRotate(5*ScaleAngle);

      /* 创建新的 Bitmap 对象 */
      Bitmap resizedBitmap =
      Bitmap.createBitmap
      (mySourceBmp, 0, 0, widthOrig, heightOrig, matrix, true);

      /**/
      BitmapDrawable myNewBitmapDrawable =
      new BitmapDrawable(resizedBitmap);

      mImageView1.setImageDrawable(myNewBitmapDrawable);
      mTextView1.setText(Integer.toString(5*ScaleAngle));
   }
});

/* 向右旋转按钮 */
```



```
        mButton2.setOnClickListener(new Button.OnClickListener()
        {
          @Override
          public void onClick(View v)
          {
            // TODO Auto-generated method stub
            ScaleAngle++;
            if(ScaleAngle>5)
            {
              ScaleAngle = 5;
            }

            /* ScaleTimes=1，维持 1:1 的宽高比例*/
            int newWidth = widthOrig * ScaleTimes;
            int newHeight = heightOrig * ScaleTimes;

            /*  计算旋转的 Matrix 比例  */
            float scaleWidth = ((float) newWidth) / widthOrig;
            float scaleHeight = ((float) newHeight) / heightOrig;

            Matrix matrix = new Matrix();
            /*  使用 Matrix.postScale 设置维度  */
            matrix.postScale(scaleWidth, scaleHeight);

            /*  使用 Matrix.postRotate 方法旋转 Bitmap*/
            //matrix.postRotate(5*ScaleAngle);
            matrix.setRotate(5*ScaleAngle);

            /*  创建新的 Bitmap 对象  */
            Bitmap resizedBitmap =
            Bitmap.createBitmap
            (mySourceBmp, 0, 0, widthOrig, heightOrig, matrix, true);

            /**/
            BitmapDrawable myNewBitmapDrawable =
            new BitmapDrawable(resizedBitmap);

            mImageView1.setImageDrawable(myNewBitmapDrawable);
            mTextView1.setText(Integer.toString(5*ScaleAngle));
          }
        });
      }
    }
```

至此，整个实例介绍完毕。执行后将显示 1 幅图片和 2 个按钮，如图 7-27 所示；分别单击“左旋转”和“右旋转”按钮后，会实现对图片旋转处理，如图 7-28 所示。



图 7-27　初始效果



图 7-28　左旋转后的效果

7.9　动态添加/删除Spinner菜单

通过本书前面示例的学习，相信读者已经大致掌握了 Spinner 的自定义菜单、交互事件的设计方法，但如果要动态增减 Spinner 下拉菜单的选项，就必须利用 ArrayList（动态数组）的依赖性来完成。

在本节的内容中，将通过一个实例的实现过程，来讲解使用 Bitmap 和 Matrix 实现图片旋转的具体流程。本实例保存在“光盘:\daima\7\tianjia”中，在本实例中，将设计一个 EditText，当 User 输入了新的文字，在点击“添加”按钮的同时，就会将输入的值添加 Spinner 至下拉菜单的最后一项，接着 Spinner 会停留在刚添加好的选项上；当点击“删除”按钮，则删除选择的 Spinner 选项，常应用于未知 Spinner 选项数量的 To-Do List 或添加维护数据等。

本实例的具体实现流程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“tianjia”的工程文件。

第 2 步：编写布局文件 main.xml 实现整体布局，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical"
  android:layout_width="fill_parent"
  android:layout_height="fill_parent"
  android:background="@drawable/white"
  >
  <TextView
  android:id="@+id/myTextView"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  android:text="@string/title"
  android:textColor="@drawable/black"
  >
  </TextView>
  <EditText
  android:id="@+id/myEditText"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
```



```
        >
        </EditText>
        <Button
        android:id="@+id/myButton_add"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="添加">
        </Button>
        <Button
        android:id="@+id/myButton_remove"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="删除">
        </Button>
        <Spinner
        android:id="@+id/mySpinner"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        >
        </Spinner>
    </LinearLayout>
```

通过上述代码，在页面中分别插入了 1 个 TextView 文本框，1 个 Button 按钮，1 个 EditText 编辑框。

第 3 步：在 string.xml 添加程序中要使用的字符串，具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <string name="title">可以动态添加、删除 Spinner 下拉菜单</string>
    <string name="app_name">你好雨夜</string>
</resources>
```

第 4 步：编写处理文件 tianjia.java，其具体实现流程如下。

1）定义数组 String[]，保存 4 个国家。

2）载入 main.xml 的布局。

3）新建 ArrayAdapter 对象并将 allCountries 传入。

4）用 findViewById()取得对象。

5）将 ArrayAdapter 添加 Spinner 对象中。

6）添加处理：先比较添加的值是否已存在，不存在才可添加；然后将值添加至 adapter，并取得添加的值的位置，将 Spinner 选择在添加的值的位置；最后，将 myEditText 清空。

7）通过 setOnClickListener 将 myButton_remove 添加 OnClickListener：首先，删除 mySpinner 的值；然后，将 myEditText 清空；最后，将 myTextView 清空。

8）通过 setOnItemSelectedListener 将 mySpinner 添加 OnItemSelectedListener，将所选 mySpinner 的值带入 myTextView 中。

文件 tianjia.java 的具体实现代码如下所示。

```
package irdc.tianjia;

import android.app.Activity;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Spinner;
import android.widget.TextView;

import irdc.tianjia.R;

import java.util.ArrayList;
import java.util.List;

public class tianjia extends Activity
{
  private static final String[] countriesStr =
  { "美国", "日本", "韩国", "英国" };
  private TextView myTextView;
  private EditText myEditText;
  private Button myButton_add;
  private Button myButton_remove;
  private Spinner mySpinner;
  private ArrayAdapter<String> adapter;
  private List<String> allCountries;

  /** Called when the activity is first created. */
  @Override
  public void onCreate(Bundle savedInstanceState)
  {
    super.onCreate(savedInstanceState);
    /* 载入 main.xml Layout */
    setContentView(R.layout.main);

    allCountries = new ArrayList<String>();
    for (int i = 0; i < countriesStr.length; i++)
    {
      allCountries.add(countriesStr[i]);
    }

    /* 新建 ArrayAdapter 对象并将 allCountries 传入 */
    adapter = new ArrayAdapter<String>(this,
        android.R.layout.simple_spinner_item, allCountries);
```



```
adapter
        .setDropDownViewResource
          (android.R.layout.simple_spinner_dropdown_item);

/*  以 findViewById()取得对象  */
myTextView = (TextView) findViewById(R.id.myTextView);
myEditText = (EditText) findViewById(R.id.myEditText);
myButton_add = (Button) findViewById(R.id.myButton_add);
myButton_remove = (Button) findViewById(R.id.myButton_remove);
mySpinner = (Spinner) findViewById(R.id.mySpinner);

/*  将 ArrayAdapter 添加 Spinner 对象中  */
mySpinner.setAdapter(adapter);

/*  将 myButton_add 添加 OnClickListener */
myButton_add.setOnClickListener(new Button.OnClickListener()
{

    @Override
    public void onClick(View arg0)
    {
        String newCountry = myEditText.getText().toString();

        /*  先比较添加的值是否已存在，不存在才可添加  */
        for (int i = 0; i < adapter.getCount(); i++)
        {
            if (newCountry.equals(adapter.getItem(i)))
            {
                return;
            }
        }

        if (!newCountry.equals(""))
        {
            /*  将值添加至 adapter */
            adapter.add(newCountry);
            /*  取得添加的值的位置  */
            int position = adapter.getPosition(newCountry);
            /*  将 Spinner 选择在添加的值的位置  */
            mySpinner.setSelection(position);
            /*  将 myEditText 清空  */
            myEditText.setText("");
        }

    }
});
```

```
        /* 将 myButton_remove 添加 OnClickListener */
        myButton_remove.setOnClickListener(new Button.OnClickListener()
        {
            @Override
            public void onClick(View arg0)
            {
                if (mySpinner.getSelectedItem() != null)
                {
                    /* 删除 mySpinner 的值 */
                    adapter.remove(mySpinner.getSelectedItem().toString());
                    /* 将 myEditText 清空 */
                    myEditText.setText("");
                    if (adapter.getCount() == 0)
                    {
                        /* 将 myTextView 清空 */
                        myTextView.setText("");
                    }
                }
            }
        });

        /* 将 mySpinner 添加 OnItemSelectedListener */
        mySpinner.setOnItemSelectedListener
        (new Spinner.OnItemSelectedListener()
        {

            @Override
            public void onItemSelected(AdapterView<?> arg0, View arg1, int arg2,
                    long arg3)
            {
                /* 将所选 mySpinner 的值带入 myTextView 中 */
                myTextView.setText(arg0.getSelectedItem().toString());
            }

            @Override
            public void onNothingSelected(AdapterView<?> arg0)
            {
            }
        });
    }
}
```

至此，整个实例介绍完毕。执行后的初始效果如图 7-29 所示；在框中输入一个国家名，单击“添加”按钮后，会将输入的国家添加到下拉框中，如图 7-30 所示；当单击“删除”按钮后，会删除当前下拉框中显示的国家，如图 7-31 所示。





图 7-29　初始效果



图 7-30　添加一个值

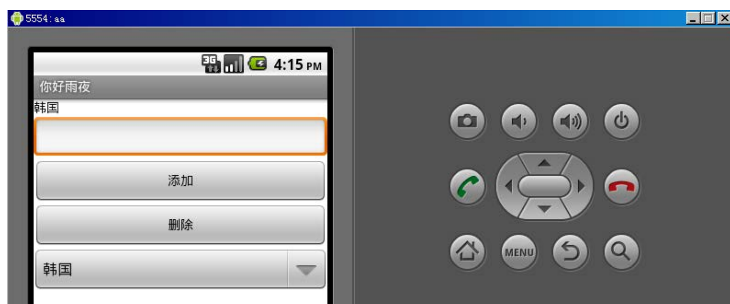


图 7-31　删除 USA 后的效果

在本实例中，setDropDownViewResource 的功能是，设置 User 点击 Spinner 后出现的下拉菜单样式，除了可以使用自设方式改变 TextView 内容之外，Android 还提供了如下两种基本的样式。

- android.R.layout.simple_spinner_item：TextView 的下拉菜单。
- ndroid.R.layout.simple_spinner_dropdown_item：除了有 TextView，右边有 radio 的下拉菜单。

第二篇　典型应用篇

第 8 章　Intent、Broadcast和Adapter详解

在前面的章节中，已经讲解了 Android 基本组件的使用知识，并且通过简单的实例程序，介绍了各个控件的使用方法。本章将对 Intent、Broadcast 和 Adapter 的基本知识进行详细剖析。

从三者的名字看它们之间没有共通的地方。而实际上，这些主题代表的是应用程序与它们的组件间绑定的粘合剂。多数平台的移动应用程序运行在它们所属的暗盒内。它们之间相互孤立，并且与系统硬件和本地组件相互作用也有着极其严格的限制。Android 应用程序也运行在暗盒内，但它们可以使用 Intent、Broadcast Receiver、Adapter、Content Provider 和 Internet 来扩展超出它们的边界。

8.1　Android体系核心之“Intent”

Intent 像消息传递机制那样使用，允许用户宣告想执行的一个动作意图，通常和一块特定的数据一起。用户可以使用 Intent 来在 Android 设备上的任何应用程序组件间相互作用，而不管它们是哪个应用程序的部分。它将一组相互独立的组件转化成一对一的相互作用的系统。

Intent 最通用的用途是启动新的 Activity，不管是显式的（通过指定类来加载）还是隐式的（通过请求在一块数据上执行的动作）。Intent 还可以用来广播消息。任何应用程序都可以注册一个 Broadcast Receiver 来监听，响应这些广播的 Intent。它让用户创建基于内部的、系统的或第三方应用程序事件的事件驱动式应用程序。

Android 通过广播 Intent 来通知系统事件，例如，网络连接状态的变化和电池充电的程度等。Android 自带的应用程序，例如，电话拨号和 SMS 管理，简单的注册组件来监听特定的广播 Intent——如来电或 SMS 消息接收，并进行相应的反应。

使用 Intent 来传播动作——甚至在同一个应用程序里，是 Android 基本的设计理念。它鼓励组件间减少耦合，允许应用程序元素无瑕疵的替换。它也为简单地扩展功能的模型提供了支持。

8.1.1　Intent的构成

要在不同的 Activity 之间传递数据，就要在 Intent 中包含相应的东西。一般来说，在数据中最基本的应该包括如下 2 点。

- Action：用来指明要实施的动作是什么，比如说 ACTION_VIEW, ACTION_EDIT 等。具体的可以查阅 Android SDK-> reference 中的 Android.content.intent 类，里面的 constants 中定义了所有的 Action。
- Data：要事实的具体的数据，一般由一个 URI 变量来表示。

如下面的代码。



```
ACTION_VIEW content://contacts/1                    //显示 identifier 为 1 的联系人的信息
ACTION_DIAL    content://contacts/1                 //给这个联系人打电话
```

除了 Action 和 data 这两个最基本的元素外，Intent 还包括一些其他的元素，具体说明如下。

- Category（类别）：这个选项指定了将要执行的这个 Action 的其他一些额外的信息，例如 LAUNCHER_CATEGORY 表示 Intent 的接受者应该在 Launcher 中作为顶级应用出现；而 ALTERNATIVE_CATEGORY 表示当前的 Intent 是一系列的可选动作中的一个，这些动作可以在同一块数据上执行。具体同样可以参考 Android SDK-> reference 中的 Android.content.intent 类。
- Type（数据类型）：显式指定 Intent 的数据类型（MIME）。一般 Intent 的数据类型能够根据数据本身进行判定，但是通过设置这个属性，可以强制采用显式指定的类型而不再进行推导。
- component（组件）：指定 Intent 的目标组件的类名称。通常 Android 会根据 Intent 中包含的其他属性的信息，比如 Action、Data/Type、Category 进行查找，最终找到一个与之匹配的目标组件。但是，如果 component 这个属性有指定的话，将直接使用它指定的组件，而不再执行上述查找过程。指定了这个属性以后，Intent 的其他所有属性都是可选的。
- Extras（附加信息）：其他所有附加信息的集合。使用 Extras 可以为组件提供扩展信息，比如，如果要执行“发送电子邮件”这个动作，可以将电子邮件的标题、正文等保存在 Extras 里，传给电子邮件发送组件。

下面是上述额外属性的几个例子。

```
//下面用于 Launch home screen
ACTION_MAIN with category CATEGORY_HOME
//下面的用于列出列表中的所有人的电话号码
ACTION_GET_CONTENT with MIME type vnd.android.cursor.item/phone
```

综上可以看出，action、data/type、category 和 extras 一起形成了一种语言，这种语言可以是 Android，可以表达出诸如“给张三打电话”之类的短语组合。

8.1.2 Intent的解析

在日常应用中，最通常以如下两种形式来使用 Intent。

- 直接 Intent：指定了 Component 属性的 Intent(调用 setComponent(ComponentName)或者 setClass(Context, Class)来指定)。通过指定具体的组件类，通知应用启动对应的组件。
- 间接 Intent：没有指定 Comonent 属性的 Intent。这些 Intent 需要包含足够的信息，这样系统才能根据这些信息，再在所有的可用组件中，确定满足此 Intent 的组件。

对于直接 Intent，Android 不需要去做解析，因为目标组件已经很明确，Android 需要解析的是那些间接 Intent，通过解析，将 Intent 映射给可以处理此 Intent 的 Activity、IntentReceiver 或 Service。

Intent 解析机制主要是通过查找已注册在 AndroidManifest.xml 中的所有<intent-filter>及其中定义的 Intent，通过 PackageManager(注：PackageManager 能够得到当前设备上所安装的 application package 的信息) 来查找能处理这个 Intent 的 component。在这个解析过程中，Android

是通过 Intent 的 Action、Type、Category 这 3 个属性来进行判断的，具体的判断方法如下。

- 如果 Intent 指明定了 Action，则目标组件的 IntentFilter 的 Action 列表中就必须包含有这个 Action，否则不能匹配。
- 如果 Intent 没有提供 Type，系统将从 Data 中得到数据类型。和 Action 一样，目标组件的数据类型列表中必须包含 Intent 的数据类型，否则不能匹配。
- 如果 Intent 中的数据不是 content 类型的 URI，而且 Intent 也没有明确指定它的 Type 类型，将根据 Intent 中数据的 Scheme（计划）进行匹配，例如“http:”或“mailto:”。同上，Intent 的 Scheme 必须出现在目标组件的 Scheme 列表中。
- 如果 Intent 指定了一个或多个 Category，这些类别必须全部出现在组建的类别列表中。比如 Intent 中包含了两个类别：LAUNCHER_CATEGORY 和 ALTERNATIVE_CATEGORY，解析得到的目标组件必须至少包含这两个类别。

下面将以 Android SDK 中自带的例子作为说明，阐述 Intent 如何定义及如何被解析。此应用可以让用户浏览便笺列表、查看每一个便笺的详细信息。其中，Manifest.xml 的代码如下。

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
  package="com.google.android.notepad">
      <application android:icon="@drawable/app_notes"
  android:label="@string/app_name">
      <provider class="NotePadProvider"
  android:authorities="com.google.provider.NotePad" />
      <activity class=".NotesList"="@string/title_notes_list">
        <intent-filter>
          <action android:value="android.intent.action.MAIN"/>
          <category android:value="android.intent.category.LAUNCHER" />
        </intent-filter>
        <intent-filter>
          <action android:value="android.intent.action.VIEW"/>
          <action android:value="android.intent.action.EDIT"/>
          <action android:value="android.intent.action.PICK"/>
          <category android:value="android.intent.category.DEFAULT" />
          <type android:value="vnd.android.cursor.dir/vnd.google.note" />
        </intent-filter>
        <intent-filter>
          <action android:value="android.intent.action.GET_CONTENT" />
          <category android:value="android.intent.category.DEFAULT" />
          <type android:value="vnd.android.cursor.item/vnd.google.note" />
        </intent-filter>
      </activity>
      <activity class=".NoteEditor"="@string/title_note">
        <intent-filter android:label="@string/resolve_edit">
          <action android:value="android.intent.action.VIEW"/>
          <action android:value="android.intent.action.EDIT"/>
          <category android:value="android.intent.category.DEFAULT" />
          <type android:value="vnd.android.cursor.item/vnd.google.note" />
```



```
            </intent-filter>
            <intent-filter>
                <action android:value="android.intent.action.INSERT"/>
                <category android:value="android.intent.category.DEFAULT" />
                <type android:value="vnd.android.cursor.dir/vnd.google.note" />
            </intent-filter>
        </activity>
        <activity class=".TitleEditor"="@string/title_edit_title"
android:theme="@android:style/Theme.Dialog">
            <intent-filter android:label="@string/resolve_title">
                <action android:value="com.google.android.notepad.action.EDIT_TITLE"/>
                <category android:value="android.intent.category.DEFAULT" />
                <category android:value="android.intent.category.ALTERNATIVE" />
                <category android:value="android.intent.category.SELECTED_ALTERNATIVE"/>
                <type android:value="vnd.android.cursor.item/vnd.google.note" />
            </intent-filter>
        </activity>
</application>
</manifest>
```

在上述代码中，一共有 3 个 Activity。

1．第一个Activity

第一个 Activity 是 com.google.android.notepad.NotesList，它是应用的主入口，提供了 3 个功能，分别由 3 个 intent-filter 进行描述。

1）第一个是进入便笺应用的顶级入口（action 为 android.app.action.MAIN）。类型为 android.app.category.LAUNCHER 表明这个 Activity 将在 Launcher 中列出。

2）第二个是当 Type 为 vnd.android.cursor.dir/vnd.google.note（保存便笺记录的目录） 时，可以查看可用的便笺（action 为 android.app.action.VIEW），或者让用户选择一个便笺并返回给调用者（action 为 android.app.action.PICK）。

3）第三个是当 Type 为 vnd.android.cursor.item/vnd.google.note 时，返回给调用者一个用户选择的便笺（action 为 android.app.action.GET_CONTENT），而用户却不需要知道便笺是从哪里读取的。有了这些功能，下面的 Intent 就会被解析到 NotesList 这个 activity。

- action=android.app.action.MAIN：与此 Intent 匹配的 Activity，将会被当做进入应用的顶级入口。
- action=android.app.action.MAIN, category=android.app.category.LAUNCHER：这是目前 Launcher 实际使用的 Intent，用于生成 Launcher 的顶级列表。
- action=android.app.action.VIEW data=content://com.google.provider.NotePad/notes：显示 content://com.google.provider.NotePad/notes 下的所有便笺的列表，使用者可以遍历列表，并且察看某便笺的详细信息。
- action=android.app.action.PICK data=content://com.google.provider.NotePad/notes：显示 content://com.google.provider.NotePad/notes 下的便笺列表，让用户可以在列表中选择一个，然后将选择的便笺的 URL 返回给调用者。

- action=android.app.action.GET_CONTENT type=vnd.android.cursor.item/vnd.google.note：和上面的 action 为 pick 的 Intent 类似，不同的是这个 Intent 允许调用者（在这里指要调用 NotesList 的某个 Activity）指定它们需要返回的数据类型，系统会根据这个数据类型查找合适的 Activity（在这里系统会找到 NotesList 这个 Activity），供用户选择便笺。

2．第二个Activity

第二个 Activity 是 com.google.android.notepad.NoteEditor，它为用户显示一条便笺，并且允许用户修改这个便笺。它定义了两个 intent-filter，所以具备了如下 2 个功能。

1）当数据类型为 vnd.android.cursor.item/vnd.google.note 时，允许用户查看和修改一个便签（action 为 android.app.action.VIEW 和 android.app.action.EDIT）。

2）当数据类型为 vnd.android.cursor.dir/vnd.google.note，为调用者显示一个新建便笺的界面，并将新建的便笺插入到便笺列表中（action 为 android.app. action.INSERT）。

有了上述两个功能，下面的 Intent 就会被解析到 NoteEditor 这个 Activity。

- action=android.app.action.VIEW data=content://com.google.provider.NotePad/notes/{ID}：向用户显示标识为 ID 的便笺。
- action=android.app.action.EDIT data=content://com.google.provider.NotePad/notes/{ID}：允许用户编辑标识为 ID 的便笺。
- action=android.app.action.INSERT data=content://com.google.provider.NotePad/notes：在“content://com.google.provider.NotePad/notes”这个便笺列表中创建一个新的空便笺，并允许用户编辑这个便签。当用户保存这个便笺后，这个新便笺的 URI 将会返回给调用者。

3．第三个Activity

最后一个 Activity 是 com.google.android.notepad.TitleEditor，它允许用户编辑便笺的标题。它可以被实现为一个应用可以直接调用（在 Intent 中明确设置 component 属性）的类，不过在此将为用户提供一个在现有的数据上发布可选操作的方法。在这个 Activity 的唯一的 intent-filter 中，拥有一个私有的 action：com.google.android.notepad.action.EDIT_TITLE，表明允许用户编辑便笺的标题。和前面的 view 和 edit 动作一样，调用这个 Intent 的时候，也必须指定具体的便笺（type 为 vnd.android.cursor.item/vnd.google.note）。不同的是，这里显示和编辑的只是便笺数据中的标题。

除了支持默认类别（android.intent.category.DEFAULT），标题编辑器还支持另外两个标准类别：android.intent.category.ALTERNATIVE 和 android.intent.category.SELECTED_ALTERNATIVE。实现了这两个类别之后，其他 Activity 就可以调用 queryIntentActivityOptions(ComponentName, Intent[], Intent, int) 查询这个 Activity 提供的 action，而不需要了解它的具体实现，或者调用 addIntentOptions(int, int, ComponentName, Intent[], Intent, int, Menu.Item[])建立动态菜单。需要说明的是，在这个 intent-filter 中有一个明确的名称（通过 android:label= "@string/resolve_title" 指定），在用户浏览数据的时候，如果这个 Activity 是数据的一个可选操作，指定明确的名称可以为用户提供一个更好的控制界面。

有了这个功能，下面的 Intent 就会被解析到 TitleEditor 这个 Activity。



```
//显示并且允许用户编辑标识为 ID 的便笺的标题
action=com.google.android.notepad.action.EDIT_TITLE
data=content://com.google.provider.NotePad/notes/{ID}
```

8.1.3　Intent的基本用法

理解 Intent 的关键之一是理解清楚 Intent 的两种基本用法：一种是显式的 Intent，即在构造 Intent 对象时就指定接收者，这种方式与普通的函数调用类似，只是复用的粒度有所差别；另一种是隐式的 Intent，即 Intent 的发送者在构造 Intent 对象时，并不知道也不关心接收者是谁，这种方式与函数调用差别比较大，有利于降低发送者和接收者之间的耦合。另外 Intent 除了发送外，还可用于广播，这些都将在后文进行详细讲述。

1．显式的Intent(Explicit Intent)的用法

概括起来，显式的 Intent 的常用用法如下。

（1）同一个应用程序中的 Activity 切换

通常一个应用程序中需要多个 UI 屏幕，也就需要多个 Activity 类，并且在这些 Activity 之间进行切换，这种切换就是通过 Intent 机制来实现的。

在同一个应用程序中切换 Activity 时，通常都会知道要启动的 Activity 具体是哪一个，因此常用显式的 Intent 来实现。例如，有一个非常简单的应用程序 SimpleIntentTest，它包括两个 UI 屏幕，也就是两个 Activity——SimpleIntentTest 类和 TestActivity 类，SimpleIntentTest 类有一个按钮用来启动 TestActivity，程序的运行效果如图 8-1 所示。



图 8-1　运行效果

程序的代码非常简单，其中 SimpleIntentTest 类的源代码如下。

```
package com.tope.samples.intent.simple;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
public class SimpleIntentTest extends Activity implements View.OnClickListener{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button startBtn = (Button)findViewById(R.id.start_activity);
        startBtn.setOnClickListener(this);
    }
    public void onClick(View v) {
```

```
            switch (v.getId()) {
            case R.id.start_activity:
                Intent intent = new Intent(this, TestActivity.class);
                startActivity(intent);
                break;
            default:
                break;
            }
        }
    }
```

在上述代码中，主要是为"Start Activity"按钮添加了OnClickListener，使得按钮被单击时执行onClick()方法，onClick()方法中则利用了Intent机制来启动TestActivity，关键的代码是如下两行。

```
Intent intent = new Intent(this, TestActivity.class);
startActivity(intent);
```

此时定义Intent对象时所用到的是Intent的构造函数之一。

```
Intent(Context packageContext, Class<?> cls)
```

这两个参数分别指定Context和Class，由于将Class设置为TestActivity.class，这样便显式地指定了TestActivity类作为该Intent的接收者，通过后面的startActivity()方法便可启动TestActivity。

TestActivity的代码更为简单，只需新建TestActivity.java文件定义1个TestActivity类即可，具体代码如下所示。

```
package com.tope.samples.intent.simple;
import android.app.Activity;
import android.os.Bundle;
public class TestActivity extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.test_activity);
    }
}
```

由此可以看出，TestActivity仅仅是调用setContentView来显示test_activity.xml中的内容而已。经过上述操作，还是不能达到利用SimpleIntentTest启动TestActivity的目的，并且会出现Exception，导致程序退出。解决方法是，在AndroidManifest.xml中增加TestActivity的声明，文件AndroidManifest.xml的具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
```



```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.tope.samples"
        android:versionCode="1"
        android:versionName="1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".SimpleIntentTest"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <!--增加 TestActivity 的声明-->
        <activity android:name=".TestActivity"/>
    </application>
    <uses-sdk android:minSdkVersion="3" />
</manifest>
```

由此可以看出，Intent 机制即使在程序内部且显式指定接收者，也还是需要在 AndroidManifest.xml 中声明 TestActivity。这个过程并不像一个简单的函数调用，显式的 Intent 也同样经过了 Android 应用程序框架所提供的支持，从满足条件的 Activity 中进行选择，如果不在 AndroidManifest.xml 中进行声明，则 Android 应用程序框架找不到所需要的 Activity。

（2）不同应用程序之间的 Activity 切换

在前面的例子中，是在同一应用程序中进行 Activity 的切换，那么在不同的应用程序中，是否也能这么做呢，答案是肯定的，不过对应的代码要稍做修改。本例中需要两个应用程序，可利用上例中的 SimpleIntentTest 作为其中之一，另外还需要写一个新的程序，来调用 SimpleIntentTest 应用程序中的 TestActivity。

先新建程序 CrossIntentTest（注意不是新建一个类，如果是 Eclipse 环境，选择“File”→“New”→“Project”新建工程），其中只有 1 个 Activity，其源代码与 SimpleIntentTest.java 类似。

```
package com.tope.samples.intent.cross;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
public class CrossIntentTest extends Activity
    implements View.OnClickListener{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
```

```
            setContentView(R.layout.main);
            Button startBtn = (Button)findViewById(R.id.start_activity);
            startBtn.setOnClickListener(this);
        }
        public void onClick(View v) {
            switch (v.getId()) {
            case R.id.start_activity:
                Intent intent = new Intent();
                intent.setClassName("com.tope.samples.intent.simple",
                        "com.tope.samples.intent.simple.TestActivity");
                startActivity(intent);
                break;
            default:
                break;
            }
        }
    }
```

由此可以看出它与 SimpleIntentTest 的不同之处在于初始化 Intent 对象的过程。

```
Intent intent = new Intent();
intent.setClassName("com.tope.samples.intent.simple",
        "com.tope.samples.intent.simple.TestActivity");
startActivity(intent);
```

此处采用了 Intent 最简单的不带参数的构造函数，然后通过 setClassName()函数来指定要启动哪个包中的哪个 Activity，而不是像上例中的通过 Intent(Context packageContext, Class<?> cls)这个构造函数来初始化 Intent 对象，这是因为，要启动的 TestActivity 与 CrossIntentTest 不在同一个包中，要指定 Class 参数比较麻烦，所以通常启动不同程序的 Activity 时便采用上面的 setClassName()的方式。除此之外，也可以利用 Android 提供的类似的 setComponent()方法，具体使用方法请参考 Android SDK 的文档。

另外还需要修改 SimpleIntentTest 程序中的 AndroidManifest.xml 文件，为 TestActivity 的声明添加 Intent Filter，即将原来的：

```
<activity android:name=".TestActivity"/>
```

修改为

```
<activity android:name=".TestActivity">
    <intent-filter>
        <action android:name="android.intent.action.DEFAULT" />
    </intent-filter>
</activity>
```

对于不同应用之间的 Activity 的切换，这里需要在 Intent Filter 中设置至少一个 Action，否则其他的应用将没有权限调用这个 Activity。这里大家开始接触 Intent Filter 和



Action 这些概念了，读者应该可以感觉到，设置 Intent Filter 和 Action 主要的目的，是为了让其他需要调用这个 Activity 的程序能够顺利地调用它。除了 Action 之外，Intent Filter 还可以设置 Category、Data 等，用来更加精确地匹配 Intent 与 Activity，这在后文将有详细介绍。

只需程序的具体运行效果，基本和前面的实例类似，这里就不再重复了。

2．隐式Intent(Implicit Intent)

如果 Intent 机制仅仅提供上面的显式 Intent 用法，则这种相对复杂的机制似乎意义并不是很大。确实，Intent 机制更重要的作用在于下面这种隐式的 Intent，即 Intent 的发送者不指定接收者，很可能不知道也不关心接收者是谁，而由 Android 框架去寻找最匹配的接收者。

（1）最简单的隐式 Intent

在此将从最简单的例子开始讲解，例如，有 1 个 ImplicitIntentTest 程序，它用于启动 Android 自带的打电话功能的 Dialer 程序。ImplicitIntentTest 程序只包含一个 Java 源文件 ImplicitIntentTest.java，具体代码如下所示。

```
package com.tope.samples.intent.implicit;
import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
public class ImplicitIntentTest extends Activity
    implements View.OnClickListener{
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button startBtn = (Button)findViewById(R.id.dial);
        startBtn.setOnClickListener(this);
    }
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.dial:
            Intent intent = new Intent(Intent.ACTION_DIAL);
            startActivity(intent);
            break;
        default:
            break;
        }
    }
}
```

上述代码是在 Intent 的使用上，与前面示例中的使用方式有很大的不同，即根本不指定

接收者，初始化 Intent 对象时，只是传入参数，设定 Action 为 Intent.ACTION_DIAL。

```
Intent intent = new Intent(Intent.ACTION_DIAL);
startActivity(intent);
```

这里使用的构造函数的原型如下。

```
Intent(String action);
```

有关 Action 的作用，本书前面的内容中已经做了简单的介绍，在此读者可以将其理解为描述这个 Intent 的一种方式，这种使用方式看上去比较奇怪，Intent 的发送者只是指定了 Action 为 Intent.ACTION_DIAL。

（2）增加一个接收者

实际上接收者如果希望能够接收某些 Intent，需要像前面例子中的一样，通过在 AndroidManifest.xml 中增加 Activity 的声明，并设置对应的 Intent Filter 和 Action，才能被 Android 的应用程序框架所匹配。为了证明这一点，修改前面 SimpleIntentTest 程序中的 AndroidManifest.xml 文件，将 TestActivity 的声明部分做如下修改。

```
<activity android:name=".TestActivity">
        <intent-filter>
            <action android:name="android.intent.action.DEFAULT" />
            <action android:name="android.intent.action.DIAL" />
            <category android:name="android.intent.category.DEFAULT" />
        </intent-filter>
</activity>
```

修改完之后注意要重新安装 SimpleIntentTest 程序的 apk 包，运行后可以发现：用户可以选择 Dialer 或者 SimpleIntentTest 程序来完成 Intent.ACTION_DIAL，也就是说，针对 Intent.ACTION_DIAL，Android 框架找到了两个符合条件的 Activity，因此它将这两个 Activity 分别列出，供用户选择。

究竟是怎么做到这一点的呢？答案是仅仅在 SimpleIntentTest 程序的 AndroidManifest.xml 文件中增加了下面的两行代码。

```
<action android:name="android.intent.action.DIAL" />
<category android:name="android.intent.category.DEFAULT" />
```

这两行修改了原来的 Intent Filter，这样这个 Activity 才能够接收到用户发送的 Intent。通过这个改动及其作用，可以进一步理解隐式 Intent，Intent Filter 及 Action，Category 等概念。Intent 发送者设定 Action 来说明将要进行的动作，而 Intent 的接收者在 AndroidManifest.xml 文件中通过设定 Intent Filter 来声明自己能接收哪些 Intent。

8.1.4 Intent和Activity

在 Android 中，Intent 和 Activity 之间是直接相互操作的。Intent 的最常用的用途是绑定应用程序组件。Intent 用来在应用程序的 Activity 间启动、停止和传输。



打开应用程序中不同的画面（Activity），调用 startActivity，传入一个 Intent，如下面的片段所示。

```
startActivity(myIntent);
```

Intent 既可以显式地指定类去打开，也可以包含目标需要执行的动作。在后者的情况下，运行时会选择 Activity 去打开，使用一个熟知的处理过程——“Intent 解析”。

startActivity 方法查找、启动与 Intent 最匹配的单一 Activity。

当使用 startActivity 时，新启动的 Activity 结束使应用程序不会接收到任何通知。为了追踪打开画面的反馈，使用 startActivityForResult 方法，在后面会描述更多细节。

1．显式启动新的Activity

经过前面知识的学习，相信读者已经了解到应用程序由很多个内部相互联系的屏幕——Activity 组成，这些 Activity 必须包含在应用程序的 manifest 中。为了连接它们，可能想要显式地指定打开哪个 Activity。

为了显式地选择一个 Activity 类来启动，需要创建一个新的 Intent，指定当前应用程序的上下文和要启动的 Activity 的类。然后传递这个 Intent 给 startActivity，例如下面的代码：

```
Intent intent = new Intent(MyActivity.this, MyOtherActivity.class);
startActivity(intent);
```

在调用 startActivity 之后，新的 Activity（在这个例子里，是 MyOtherActivity）将被创建，并变成可见和活跃状态，移到 Activity 栈的最顶端。

代码调用新 Activity 的 finish 方法会关闭它，并从栈中移除。可变通的地方，用户可以通过设备的【Back】按钮导航到先前的 Activity。

2．隐式Intent和运行时绑定

隐式 Intent 是一种让匿名应用程序组件服务动作请求的机制。当创建一个新的隐式 Intent 时，用户指定要执行的动作，作为可选项，可以提供这个动作所需的数据。

当使用这个新的隐式 Intent 来启动 Activity 时，Android 会在运行时解析它，找到最适合在指定的数据类型上执行动作的类。这意味着，可以创建使用其他应用程序的工程，而不需要提前精确地知道会借用哪个应用程序的功能。例如，如果想让用户在应用程序里打电话，与其实现一个新的拨号，不如使用一个隐式的 Intent 来请求一个在一个电话号码（URI 表示）上的动作（拨一个号码），例如使用下面的代码。

```
if (somethingWeird && itDontLookGood)
{
Intent intent = new Intent(Intent.ACTION_DIAL, Uri.parse(“tel:555-2368”));
startActivity(intent);
}
```

Android 解析这个 Intent 并启动一个提供了能在一个号码上执行拨号动作的 Activity，在这里，是拨号 Activity。一些本地的应用程序提供了在特定数据上执行动作的组件。第三方应用程序，也可以注册来支持新的动作或为本地动作提供一种替代的方法。

8.1.5 Linkify简介

Linkify 是一个辅助类，通过 RegEx 样式匹配，自动地在 TextView 类（和继承的类）中创建超链接。符合特定的 RegEx 样式的文本会被转变成可单击的超链接，这些超链接隐式地调用 startActivity(new Intent(Intent.ACTION_VIEW,uri))，符合的文本会作为目标 URI。用户可以指定任意的字符串样式为链接；方便地，Linkify 类提供了预置的通用内容类型（如电话号码和 E-mail、Web 地址）。

1．本地的链接类型

Linkify.addLinks 静态方法接受一个 View 来制作链接，还包括一个或多个支持的默认内容类型的位结果。Linkify 类提供了一些内容类型：WEB_URLS、EMAIL_ADDRESSES、PHONE_NUMBERS 和 ALL。

例如，下面的代码显示了如何为 TextView 制作链接显示 Web 和 E-mail 地址为超链接。当单击时，它们会相应地打开浏览器或 E-mail 应用程序。

```
TextView textView = (TextView)findViewById(R.id.myTextView);
Linkify.addLinks(textView, Linkify.WEB_URLS|Linkify.EMAIL_ADDRESSES);
```

读者可以在 layout 资源中，通过使用 oid:autoLink 特性来为 View 制作链接。它支持一个或多个（用|分割）自定义的值：none、web、email、phone 或 all。接下来的 XML 片段显示了如何为电话号码和 E-mail 地址添加超链接。

```
<TextView
android:layout_width=”fill_parent”
android:layout_height=”fill_parent”
android:text=”@string/linkify_me”
android:autoLink=”phone|email”
/>
```

2．使用Match Filter

在定义的 MatchFilter 中实现 acceptMatch 方法，来为 RegEx 样式匹配添加额外的条件。当一个潜在的匹配发现时，acceptMatch 被触发，匹配的开始点和结束点（包括被查找的整个文本）以参数的形式传入。例如，下面的代码显示了一个 MatchFilter 的实现，它取消任何之前是一个“!”的匹配。

```
class MyMatchFilter implements MatchFilter {
public boolean acceptMatch(CharSequence s, int start, int end) {
return (start == 0 || s.charAt(start-1) != ‘!’);
}
}
```

3．使用Transform Filter

Transform Filter 为格式化文本字符串提供了更大的自由度，允许用户修改由链接文本自动生成的隐式 URI。减少链接文本和目标 URI 的耦合，能更加自由地决定如何显示数据字符串给用户。



使用 Transform Filter，在定义的 TransformFilter 中实现 transformUrl 方法。当 Linkify 找到正确的匹配后，它会调用 transformUrl，传入使用的 RegEx 样式和它创建的默认 URI 字符串。可以修改匹配的字符串，然后返回一个适合给其他 Android 应用程序“看”的 URI。例如下面的 TransformFilter 实现将匹配的文本转换成小写的 URI。

```
class MyTransformFilter implements TransformFilter {
public String transformUrl(Matcher match, String url) {
return url.toLowerCase();
}
}
```

4．创建自定义的链接字符串

为了定义自己的链接字符串，需要创建一个 RegEx 样式来匹配文本，进而显示成超链接。和本地类型一样，通过调用 Linkify.addLinks 来指定目标 View，但这次，传入的是新的 RegEx 样式。还可以传入一个前缀，当链接单击时，它会添加到目标 URI 上。

例如在下面的代码中显示了一个 View 链接到由 Android Content Provider（下一章创建）提供的地震数据。与包含所有的情况相比，链接样式能匹配任何以“quake”开头后跟一个数字的文本。在 Intent 被触发前，内容会被添加到 URI 上。

```
int flags = Pattern.CASE_INSENSITIVE;
Pattern p = Pattern.compile(“\\bquake[0−9]*\\b”, flags);
Linkify.addLinks(myTextView, p, “content://com.paad.earthquake/earthquakes/”);
```

Linkify 还支持 TranformFilter、MatchFilter 接口。它们提供一些对目标 URI 的额外控制和定义匹配字符串，它们的使用如以下代码所示。

```
Linkify.addLinks(myTextView, pattern, prefixWith, new MyMatchFilter(), new MyTransformFilter());
```

8.2　Activity的返回值和本地动作

使用 startActivity 方式启动的 Activity 和它的父 Activity 无关，当它关闭时也不会提供任何反馈。同理，用户可以启动一个 Activity 作为子 Activity，它与父 Activity 有内在的联系。当子 Activity 关闭时，它会触发父 Activity 中的一个事件处理函数。子 Activity 最适合用在一个 Activity 为其他的 Activity 提供数据（例如用户从一个列表中选择一个项目）的场合。

子 Activity 的创建和普通 Activity 的创建相同，也必须在应用程序的 manifest 中注册。任何在 manifest 中注册的 Activity 都可以用做子 Activity。

8.2.1　Activity返回值

1．启动子Activity

startActivityForResult 方法和 startActivity 方法工作很相似，但有一个很重要的差异。Intent 都是用来决定启动哪个 Activity，用户还可以传入一个请求码。这个值将在后面用来作为有返回值 Activity 的唯一 ID。例如下面的代码显示了如何启动一个子 Activity。

```
private static final int SHOW_SUBACTIVITY = 1;
Intent intent = new Intent(this, MyOtherActivity.class);
startActivityForResult(intent, SHOW_SUBACTIVITY);
```

和正常的 Activity 一样，子 Activity 可以隐式或显式启动。例如，下面的代码使用一个隐式的 Intent 来启动一个新的子 Activity 来挑选一个联系人。

```
private static final int PICK_CONTACT_SUBACTIVITY = 2;
Uri uri = Uri.parse(“content://contacts/people”);
Intent intent = new Intent(Intent.ACTION_PICK, uri);
startActivityForResult(intent, PICK_CONTACT_SUBACTIVITY);
```

2．返回值

当子 Activity 准备关闭时，在结束之前调用 setResult 来给调用的 Activity 返回一个结果。setResult 方法带两个参数：结果码和表示为 Intent 的负载值。结果码是运行子 Activity 的结果，一般是 Activity.RESULT_OK 或 Activity.RESULT_CANCELED。在一些情况下，用户会希望使用自己的响应代号来处理特定的应用程序的选择；setResult 支持任何整数值。

作为结果返回的 Intent 可以包含指向一个内容（例如，联系人，电话号码或媒体文件）的 URI 和一组用来返回额外信息的 Extra。

下面的代码片段节选自子 Activity 的 onCreate 方法，显示了怎样向调用的 Activity 返回不同的结果。

```
Button okButton = (Button) findViewById(R.id.ok_button);
okButton.setOnClickListener(new View.OnClickListener() {
public void onClick(View view)
{
Uri data = Uri.parse(“content://horses/” + selected_horse_id);
Intent result = new Intent(null, data);
result.putExtra(IS_INPUT_CORRECT, inputCorrect);
result.putExtra(SELECTED_PISTOL, selectedPistol);
setResult(RESULT_OK, result);
finish();
}
});
Button cancelButton = (Button) findViewById(R.id.cancel_button);
cancelButton.setOnClickListener(new View.OnClickListener() {
public void onClick(View view)
{
setResult(RESULT_CANCELED, null);
finish();
}
});
```

3．处理子Activity的结果

当子 Activity 关闭时，它的父 Activity 的 onActivityResult 事件处理函数被触发。重写这



个方法来处理从子 Activity 返回的结果。onActivityResult 处理器接受如下几个参数。

- 请求码：曾经用来启动子 Activity 的请求码。
- 结果码：由子 Activity 设置的，用来显示它的结果。它可以是任何整数值，但典型的值是 Activity.RESULT_OK 和 Activity.RESULT_CANCELLED。如果子 Activity 非正常关闭或在关闭时没有指定结果码，结果码都是 Activity.RESULT_CANCELED。
- 数据：一个 Intent 来打包任何返回的数据。依赖于子 Activity 的目的，它可能会包含一个代表特殊的从列表中选择的数据的 URI。可变通的，或额外的，子 Activity 可以使用“extras”机制以基础值的方式返回临时信息。

例如，下面的代码实现了一个 Activity 中的 onActivityResult 事件处理函数。

```
private static final int SHOW_SUB_ACTIVITY_ONE = 1;
private static final int SHOW_SUB_ACTIVITY_TWO = 2;
@Override
public void onActivityResult(int requestCode, int resultCode, Intent data) {
super.onActivityResult(requestCode, resultCode, data);
switch(requestCode)
{
case (SHOW_SUB_ACTIVITY_ONE) :
{
if (resultCode == Activity.RESULT_OK)
{
Uri horse = data.getData();
boolean inputCorrect = data.getBooleanExtra(IS_INPUT_CORRECT, false);
String selectedPistol = data.getStringExtra(SELECTED_PISTOL);
}
break;
}
case (SHOW_SUB_ACTIVITY_TWO) :
{
if (resultCode == Activity.RESULT_OK)
{
// TODO: Handle OK click.
}
break;
}
}
}
```

8.2.2　Android本地动作

Android 本地应用程序也使用 Intent 来启动 Activity 和子 Activity。下面简单地列出了 Intent 类中以静态字符串常量保存的本地动作。读者可以在自己的应用程序里当创建隐式 Intent 来启动 Activity 和子 Activity 时使用这些动作。

- ACTION_ANSWER：打开一个 Activity 来处理来电。目前，它是被本地的电话拨号工

具处理。

- ❑ ACTION_CALL：启动电话拨号工具，并立即用数据 URI 中的号码初始化一个呼叫。一般来说，如果可能的话，它认为是比使用 Dial_Action 好的一种方式。
- ❑ ACTION_DELETE：启动一个 Activity 来让用户删除储存在 URI 位置的数据入口。
- ❑ ACTION_DIAL：启动一个电话拨号程序，使用预置在数据 URI 中的号码来拨号。默认情况下，它是由 Android 本地的电话拨号工具处理。这个拨号工具能规范多数的号码；举个例子，tel:555-1234 和 tel:(212)555 1212 都是有效的号码。
- ❑ ACTION_EDIT：请求一个 Activity 来编辑 URI 处的数据。
- ❑ ACTION_INSERT：打开一个能在数据域的特定游标处插入新项目的 Activity。当以子 Activity 方式调用时，它必须返回新插入项目的 URI。
- ❑ ACTION_PICK：启动一个子 Activity 来让用户从 URI 数据处挑选一个项目。当关闭时，它必须返回指向被挑选项目的 URI。启动的 Activity 取决于要挑选的数据；例如，传入 content://contacts/people 会引发本地的联系人列表。
- ❑ ACTION_SEARCH：启动一个 UI 来执行搜索。在 Intent 的数据包里使用 SearchManager.QUERY 键值来提供搜索内容的字符串。
- ❑ ACTION_SENDTO：通常启动一个 Activity 来给 URI 中的指定联系人发送一个消息。
- ❑ ACTION_SEND：启动一个 Activity 来发送特定的数据（接收者经由解析 Activity 来选择）。使用 setType 来设置 Intent 的类型为传输数据的 mime 类型。数据本身依赖于类型使用 EXTRA_TEXT 或 EXTRA_STREAM 来储存。在 E-mail 的情况下，Android 本地应用程序还可以接受使用 EXTRA_EMAIL, EXTRA_CC, EXTRA_BCC, 和 EXTRA_SUBJECT 键值的 extras。
- ❑ ACTION_VIEW：最通用的动作。View 动作要求 Intent URI 中的数据以最合理的方式显示。不同的应用程序将处理 View 请求，依赖于 URI 中的数据。一般的，http：地址会在浏览器中打开；tel：地址会在拨号工具中打开并呼叫号码；geo：地址会在地图应用程序中显示；联系人内容会在联系人管理器中显示。
- ❑ ACTION_WEB_SEARCH：打开一个 Activity，执行基于数据 URI 中文本的网页搜索。

和这些 Activity 动作一样，Android 还包括大量的 Broadcast 动作，用来创建 Intent 将系统消息通知给应用程序。这些 Broadcast 动作将在这章稍后部分描述。

8.2.3 Intent Filter响应隐式Intent

如果一个 Intent 请求在一片数据上执行一个动作，Android 如何知道哪个应用程序（和组件）能用来响应这个请求呢？Intent Filter 就是用来注册 Activity、Service 和 Broadcast Receiver 具有能在某种数据上执行一个动作的能力。

使用 Intent Filter 时，应用程序组件会告诉 Android，它们能为其他程序的组件的动作请求提供服务，包括同一个程序的组件、本地的或第三方的应用程序。

为了注册一个应用程序组件为 Intent 处理者，在组件的 manifest 节点添加一个 intent-filter 标签。在 Intent Filter 节点里使用下面的标签（关联属性），就可以指定组件支持的动作、种类和数据。

- ❑ Action：使用 android:name 特性来指定对响应的动作名。动作名必须是独一无二的字



符串，所以，一个好的习惯是使用基于 Java 包的命名方式的命名系统。

- Category：android:category 属性用来指定在什么样的环境下动作才被响应。每个 Intent Filter 标签可以包含多个 category 标签。用户可以指定自定义的种类或使用 Android 提供的标准值，如下所示。
  - ALTERNATIVE：一个 Intent Filter 的用途是使用动作来帮忙填入上下文菜单。ALTERNATIVE 种类指定，在某种数据类型的项目上可以替代默认执行的动作。例如，一个联系人的默认动作时浏览它，替代的可能是去编辑或删除它。
  - SELECTED_ALTERNATIVE：与 ALTERNATIVE 类似，但 ALTERNATIVE 总是使用下面所述的 Intent 解析来指向单一的动作。SELECTED_ALTERNATIVE 在需要一个可能性列表时使用。
  - BROWSABLE：指定在浏览器中的动作。当 Intent 在浏览器中被引发，都会被指定成 BROWSABLE 种类。
  - DEFAULT：设置这个种类来让组件成为 Intent Filter 中定义的 data 的默认动作。这对使用显式 Intent 启动的 Activity 来说也是必要的。
  - GADGET：通过设置 GADGET 种类，你可以指定这个 Activity 可以嵌入到其他的 Activity 来允许。
  - HOMEHOME Activity：设备启动（登录屏幕）时显示的第一个 Activity。通过指定 Intent Filter 为 HOME 种类而不指定动作，你正在将其设为本地 home 画面的替代。
  - LAUNCHER：使用这个种类来让一个 Activity 作为应用程序的启动项。
  - datadata 标签允许指定组件能作用的数据的匹配；如果你的组件能处理多个的话，你可以包含多个条件。你可以使用下面属性的任意组合来指定组件支持的数据：
  - android:host：指定一个有效的主机名（例如，com.google）。
  - ❑ android:mimetype：允许设定组件能处理的数据类型。例如，<type android:value=" vnd.android.cursor.dir/*" />能匹配任何 Android 游标。
  - android:path：有效的 URI 路径值（例如，/transport/boats/）。
  - android:port：特定主机上的有效端口。
  - android:scheme：需要一个特殊的图示（例如，content 或 http）。

接下来的代码片段显示了如何配置 Activity 的 Intent Filter，使其以在特定数据下的默认的或可替代的动作的身份来执行 SHOW_DAMAGE 动作。（创建地震内容将在下一章节讲解。）

```
<activity android:name=”.EarthquakeDamageViewer”
android:label=”View Damage”>
<intent-filter>
<action
android:name=”com.paad.earthquake.intent.action.SHOW_DAMAGE”>
</action>
<category android:name=”android.intent.category.DEFAULT”/>
<category
android:name=”android.intent.category.ALTERNATIVE_SELECTED”
/>
```

```
<data android:mimeType="vnd.earthquake.cursor.item/*"/>
</intent-filter>
</activity>
```

8.2.4 Android解析Intent Filter

在运行 Android 程序时支持匿名机制，当解析一个隐式 Intent 到一个应用程序组件时，这种隐式机制变得十分重要。当使用 startActivity 时，隐式 Intent 解析到一个单一的 Activity。如果存在多个 Activity 都有能力在特定的数据上执行给定的动作，Android 就会从中选择最好的进行启动。

决定哪个 Activity 来运行的过程称为 Intent 解析。Intent 解析的目的是通过下面的过程找到可能匹配得最好的 Intent Filter。

第 1 步：Android 把安装的包中可获得的 Intent Filter 放到一个列表中。

第 2 步：动作和与正在解析的 Intent 的种类不关联的 Intent Filter 会从列表中删除。

1）动作匹配指 Intent Filter 包含特定的动作或没有指定的动作。一个 Intent Filter 有一个或多个定义的动作，如果没有任何一个能与 Intent 指定的动作匹配的话，这个 Intent Filter 算做是动作匹配检查失败。

2）种类匹配更为严格。Intent Filter 必须包含所有在解析的 Intent 中定义的种类。一个没有特定种类的 Intent Filter 只能与没有种类的 Intent 匹配。

第 3 步：Intent 的数据 URI 中的部分会与 Intent Filter 中的 data 标签比较。如果 Intent Filter 定义 scheme，host/authority，path 或 mimetype，这些值都会与 Intent 的 URI 比较。任何不匹配都会导致 Intent Filter 从列表中删除。

如果没有指定 data 值的 Intent Filter 会和所有的 Intent 数据匹配。

1）mimetype 是正在匹配的数据的数据类型。当匹配数据类型时，用户可以使用通配符来匹配子类型（例如，earthquakes/*）。如果 Intent Filter 指定一个数据类型，它必须与 Intent 匹配；没有指定数据则全部匹配。

2）scheme 是 URI 部分的协议——例如，http:，mailto:，tel:。

3）host-name 或“data authority”是介于 URI 中 scheme 和 path 之间的部分。匹配主机名时，Intent Filter 的 scheme 也必须通过匹配。

4）数据 path 是紧接在“data authority”的后面，例如，/ig。当 path 只在 scheme 和 host-name 部分都匹配的情况下才匹配。

第 4 步：如果这个过程中有多于一个组件解析出来的话，它们会以优先度来排序，可以在 Intent Filter 的节点里添加一个可选的标签。最高等级的组件会返回。

Android 本地的应用程序组件和第三方应用程序一样，都是 Intent 解析过程中的一部分。它们没有更高的优先度，可以被新的 Activity 完全代替，这些新的 Activity 宣告自己的 Intent Filter 能响应相同的动作请求。

8.2.5 隐式Intent响应与责任传递

1. Intent Filter匹配后的响应

当一个应用程序的组件通过一个隐式 Intent 启动后，它需要找到它要执行的动作以及执



行动作需要的数据。调用 getIntent 方法（一般在 onCreate 方法里）来释放 Intent，从而启动一个组件，格式如下。

```
@Override
public void onCreate(Bundle icicle) {
super.onCreate(icicle);
setContentView(R.layout.main);
Intent intent = getIntent();
}
```

使用 getData 方法和 getAction 方法来查找 Intent 的数据和动作。使用类型安全的 get<type>Extra 方法来释放临时信息包中的额外信息，代码如下。

```
String action = intent.getAction();
Uri data = intent.getData();
```

2．传递责任

可以使用 startNextMatchingActivity 方法来将动作处理的责任传递给下一个最匹配的应用程序组件，如以下片段所示。

```
Intent intent = getIntent();
if (isAfterMidnight)
startNextMatchingActivity(intent);
```

这允许可以为组件添加额外的条件，限制其超出 Intent Filter 能力以外的使用。

实际上在某些情况下，组件可能希望在传递 Intent 到本地处理器之前能执行一些处理，或者提供用户一个选择。

8.3 用Intent来广播一个事件

作为一种系统级消息传递的机制，Intent 有能力穿越进程边界传递结构化消息。到目前为止，大家已经了解了使用 Intent 来启动一个新的应用程序组件，但是，它们还可以通过 sendBroadcast 方法在组件间广播匿名消息。用户可以在应用程序中实现 Broadcast Receiver 来监听和响应这些广播的 Intent。广播 Intent 用于通知系统的监听者或应用程序事件，从而扩展了应用程序间的事件驱动编程模型。

广播 Intent 让用户的程序更加开放；通过使用 Intent 来广播事件，让第三方开发者响应事件而不需要修改用户的原始程序。在用户的应用程序里，可以监听广播的 Intent 来替换或增强本地的（或第三方的）应用程序，或者对系统变化和应用程序事件做出响应。举例来说，通过监听外来的呼叫广播，可以改变呼叫者的铃声或音量。

Android 广泛地使用广播 Intent 来广播系统事件，如电池充电变化、网络连接和来电。

8.3.1 广播事件

广播 Intent 相当简单。在用户的程序组件里，构建要广播的 Intent，使用 sendBroadcast

方法发送出去。设定 Intent 的动作、数据和种类来使 Broadcast Receiver 精确地决定它们的兴趣。在这里，Intent 动作字符串用来标识要广播的事件，因此，它必须是独一无二的标识事件的字符串。习惯的，动作字符串使用 Java 包的样子来定义，例如下面的代码。

```
public static final String NEW_LIFEFORM_DETECTED =
"com.paad.action.NEW_LIFEFORM";
```

如果想在 Intent 中包含数据，可以使用 Intent 的 data 属性来指定一个 URI；还可以包含 extras 来增加额外的本地类型值。就事件驱动模型而言，这些 extras 包等价于事件处理函数的可选参数。

例如，下面的框架代码给出了一个广播的 Intent 的基本创建，使用之前定义的动作和一些以 extras 方法储存的额外的事件信息。

```
Intent intent = new Intent(NEW_LIFEFORM_DETECTED);
intent.putExtra("lifeformName", lifeformType);
intent.putExtra("longitude", currentLongitude);
intent.putExtra("latitude", currentLatitude);
sendBroadcast(intent);
```

8.3.2　Broadcast Receiver监听广播

Broadcast Receiver 用于监听广播 Intent。为了激活一个 Broadcast Receiver，需要在代码或在程序 manifest 中注册。当注册一个 Broadcast Receiver 时，必须使用 Intent Filter 来指定要监听哪个 Intent。

为了创建一个新的 Broadcast Receiver，需要扩展 BroadcastReceiver 类，并重写 onReceive 事件处理函数，如下面的框架代码所示。

```
import android.content.BroadcastReceiver;
import android.content.Context;
import android.content.Intent;
public class MyBroadcastReceiver extends BroadcastReceiver {
@Override
public void onReceive(Context context, Intent intent) {
//TODO: React to the Intent received.
}
}
```

当广播的 Intent 与注册的接收器的 Intent Filter 匹配时，onReceive 方法会执行。onReceive 处理函数必须在 5s 内完成，否则应用程序无响应的对话框会显示。

在 Intent 广播的时候，注册有 Broadcast Receiver 的应用程序不需要正在运行。它们在有匹配的广播 Intent 时自动启动。这对于资源管理来说，是极好的，因为它允许用户创建可以被关闭或杀死的事件驱动应用程序，而此刻又以安全的方式对广播事件做出响应。

典型的，Broadcast Receiver 会更新内容、启动服务、更新 Activity 的 UI 或使用通知管理器来通知用户。5s 的执行限制确保了主进程不能，或者说不应该在 Broadcast Receiver 中直接结束。



例如,下面的代码显示了如何实现一个 Broadcast Receiver。在下一个章节，将学习如何在代码或在 manifest 中注册它。

```
public class LifeformDetectedBroadcastReceiver extends BroadcastReceiver {
public static final String BURN =
"com.paad.alien.action.BURN_IT_WITH_FIRE";
@Override
public void onReceive(Context context, Intent intent) {
// Get the lifeform details from the intent.
Uri data = intent.getData();
String type = intent.getStringExtra("type");
double lat = intent.getDoubleExtra("latitude", 0);
double lng = intent.getDoubleExtra("longitude", 0);
Location loc = new Location("gps");
loc.setLatitude(lat);
loc.setLongitude(lng);
if (type.equals("alien")) {
Intent startIntent = new Intent(BURN, data);
startIntent.putExtra("latitude", lat);
startIntent.putExtra("longitude", lng);
context.startActivity(startIntent);
}
}
}
```

1．在程序的manifest中注册

为了在程序的 manifest 中包含一个 Broadcast Receiver，通过在 application 节点增加一个 receiver 标签，并指定要注册的 Broadcast Receiver 的类名。receiver 节点需要包含一个 intent-filter 标签来指定要监听的动作字符串，如下面的 XML 片段所示。

```
<receiver android:name=".LifeformDetectedBroadcastReceiver">
<intent-filter>
<action android:name="com.paad.action.NEW_LIFEFORM"/>
</intent-filter>
</receiver>
```

Broadcast Receiver 以这种方法注册将总是处于活跃状态。

2．在代码中注册

也可以在代码中控制 Broadcast Receiver 的注册。这种做法的典型例子就是 receiver 用来在 Activity 中更新 UI 元素。一个好的习惯是当 Activity 不可见(或不活跃)时，反注册 Broadcast Receiver。

例如，下面的代码显示了如何使用一个 Intent Filter 注册 Broadcast Receiver。

```
// Create and register the broadcast receiver.
IntentFilter filter = new IntentFilter(NEW_LIFEFORM_DETECTED);
LifeformDetectedBroadcastReceiver r = new LifeformDetectedBroadcastReceiver();
```

```
registerReceiver(r, filter);
```

为了反注册一个 Broadcast Receiver，在程序上下文中使用 unregisterReceiver 方法，传入一个 Broadcast Receiver 实例，如下所示。

```
unregisterReceiver(r);
```

8.3.3 Android本地广播

Android 给许多系统服务广播 Intent。可以使用这些基于系统事件的消息来给自己的工程增添一些功能，这些事件如时区变更、数据连接状态、SMS 消息或电话呼叫。

在下面列出了一些 Intent 类中的本地动作常量；这些动作基本上用于设备状态改变的跟踪。

- ACTION_BOOT_COMPLETED：一旦设备完成启动时触发。需要 RECEIVE_BOOT_COMPLETED 权限。
- ACTION_CAMERA_BUTTON：摄像头按钮被按下时触发。
- ACTION_DATE_CHANGED 和 ACTION_TIME_CHANGED：当手动修改日期或时间时广播这两个动作。
- ACTION_GTALK_SERVICE_CONNECTED 和 ACTION_GTALK_SERVICE_DISCONNECTED：当 GTalk 连接或丢失连接时广播这两个动作。更多 GTalk 消息将在第 9 章详细讨论。
- ACTION_MEDIA_BUTTON：媒体按钮按下时触发。
- ACTION_MEDIA_EJECT：当用户选择弹出外部的储存媒体，会首先触发这个。如果用户的程序读/写到外部媒体存储器，应该监听这个事件来保存和关闭任何打开的文件句柄。
- ACTION_MEDIA_MOUNTED 和 ACTION_MEDIA_UNMOUNTED：当新的外部存储媒体成功地添加到设备或从设备移除时触发。
- ACTION_SCREEN_OFF 和 ACTION_SCREEN_ON：当屏幕打开或关闭时广播。
- ACTION_TIMEZONE_CHANGED：当电话的当前时区变更时会广播这个动作。Intent 中包含一个 ID 为 java.util.TimeZone 的时区 extras。Android 用来通知应用程序状态变更的动作的列表可参考 http://code.google.com/android/reference/android/content/ntent.html。

Android 还使用 Broadcast Receiver 来监听特定的事件如 SMS 消息接收。动作和与这些事件关联的 Intent 将在后面的章节详细讨论。

8.4 Adapter详解

Adapter 是将数据绑定到 UI 界面上的桥接类。Adapter 负责创建显示每个项目的子 View 和提供对下层数据的访问。支持 Adapter 绑定的 UI 控件必须扩展 AdapterView 抽象类。创建自己的继承自 AdapterView 的控件和创建新的 Adapter 类来绑定它们是可能的。



8.4.1　Android提供的Adapter

在大多数情况下，不需要创建自己的 Adapter。Android 提供了一系列 Adapter 来将数据绑定到 UI Widget 上。因为 Android 负责提供数据和选择用于显示每个项目的 View，所以 Adapter 能快速地修改要绑定的控件的外观和功能。下面列出了两个最有用和最通用的本地 Adapter。

1．ArrayAdapter

ArrayAdapter 是一个绑定 View 到一组对象的通用类。默认情况下，ArrayAdapter 绑定每个对象的 toString 值到在 layout 中预先定义的 TextView 控件上。可变通的，构造函数允许用户使用更加复杂的 layout 或者通过重写 getView 方法来扩展类从而使用 TextView 的替代物（如 ImageView 或嵌套的 layout）。

2．SimpleCursorAdapter

SimpleCursorAdapter 绑定 View 到 Content Provider 查询返回的游标上。指定一个 XML layout 定义，然后将数据集中的每一列的值绑定到 layout 中的一个 View 上。

8.4.2　用Adapter进行绑定

可以使用 Adapter 进行数据绑定。将 Adapter 应用到继承自 AdapterView 类上，需要调用 View 的 setAdapter 方法，传入一个 Adapter 实例，例如下面的代码。

```
ArrayList<String> myStringArray = new ArrayList<String>();
ArrayAdapter<String> myAdapterInstance;
int layoutID = android.R.layout.simple_list_item_1;
myAdapterInstance = new ArrayAdapter<String>(this, layoutID, myStringArray);
myListView.setAdapter(myAdapterInstance);
```

上述代码显示了最简单的情况，将数组中的字符串绑定到 ListView 中用于显示每个项目的简单 TextView 控件上。

8.4.3　使用ArrayAdapter和SimpleCursorAdapter

在前面的内容中，已经多次使用过了 ArrayAdapter 和 SimpleAdapter。和它们俩类似，SimpleCursorAdapter 也是集成 Adapter。ArrayAdapter 负责把一个字符串数组中的数据填充到一个 ListView 当中，而对应的 SimpleCursorAdapter 负责把 Cursor 里边的内容填充到 ListView 当中。通过 SimpleCursorAdapter 可以把数据库中一列的数据和 ListView 中一排对应起来。和前两个 Adapter 类似，要求和数据进行对应的 View 必须是 TextView 或者 ImageView。

1．使用ArrayAdapter

本节将通过一个实例的实现过程，来讲解使用 CursorAdapter 的方法。本实例对应的源代码文件保存在“光盘\daima\8\ArrayAdapterUsage”文件夹内，具体实现流程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“ArrayAdapterUsage”的工程文件。

第 2 步：编写布局文件 main.xml，插入一个 ListView 控件，实现对元素的布局处理。具

体实现代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<ListView
     android:id="@+id/list"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    />
</LinearLayout>
```

第 3 步：编写处理文件 ArrayAdapterUsage.java，实现事件处理功能，具体代码如下。

```
package com.android.Adapter.ArrayAdapter;
import android.app.Activity;
import android.os.Bundle;
import android.widget.ArrayAdapter;
import android.widget.ListView;
public class ArrayAdapterUsage extends Activity {
     ListView lv;
     String[] value = {
               "JAN","FEB","MAR","APR",
               "MAY","JUN","JUL","AUG","SEP","OCT","NOV","DEC "
     };
     /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);

        lv = (ListView) findViewById(R.id.list);

        ArrayAdapter<String> adapter = new ArrayAdapter<String>(this,android.R.layout.simple_list_item_1,
value);

        lv.setAdapter(adapter);
    }
}
```

在上述代码中，首先定义了一个 String[]，里面保存了英文中的 12 个月份。然后通过 ListView 将数组中的数据显示出来。至此整个实例介绍完毕，执行后的效果如图 8-2 所示。





图 8-2　执行效果

从图 8-2 所示的效果可以看出，当鼠标滚动在某一月份后，会以突出颜色样式显示。

2．使用SimpleCursorAdapter

SimpleCursorAdapter 允许用户绑定一个游标的列到 ListView 上，并使用自定义的 Layout（布局）显示每个项目。要实现 SimpleCursorAdapter 的创建，需要传入当前的上下文、一个 layout 资源，一个游标和两个数组；其中一个包含使用的列的名字，另一个数组包含 View 中的资源 ID，用于显示相应列的数据值。例如，通过下面的代码，介绍了如何构造一个 SimpleCursorAdapter 来显示联系人信息。

```
String uriString = “content://contacts/people/”;
Cursor myCursor = managedQuery(Uri.parse(uriString), null, null, null, null);
String[] fromColumns = new String[] {People.NUMBER, People.NAME};
int[] toLayoutIDs = new int[] { R.id.nameTextView, R.id.numberTextView};
SimpleCursorAdapter myAdapter;
myAdapter = new SimpleCursorAdapter(this,R.layout.simplecursorlayout,myCursor,fromColumns,
toLayoutIDs);
myListView.setAdapter(myAdapter);
```

注意：SimpleCursorAdapter 前面的创建选择联系人的例子中使用过。在本书后面的 Content Provider 和 Cursor 的内容中，将会有更多 SimpleCursorAdapter 的使用实例。

8.5　Internet资源

在网络连通和拥有 WebKit 浏览器的前提下，大家可能会问，在创建 Web 应用程序时，能否创建一个本地的、能够替换远程 Internet 的应用？这其实是创建胖的和瘦的客户端应用程序的问题，这也是网络服务器领域一个长远的证题创建一个而不依赖于整个 Web 的解决方案有很多益处，具体说明如下。

（1）带宽

在有限的并且昂贵的带宽约束下，静态的资源如图片、layout 和声音等都是设备上昂贵数据的消费者。通过创建一个本地的应用程序，可以限制只用于数据更新的带宽需求。

（2）缓存

移动网络存储还没有达到普遍存在的状态。使用一个基于浏览器的解决方案，部分网络连通可能导致应用程序断断续续。本地的应用程序可以由缓存数据来提供尽可能多的功能而不需要一个实时的连接。

（3）本地特征

Android 设备不仅仅是一个运行浏览器的平台；它还包括定位的服务、摄像头硬件和重力加速度计。通过创建本地的应用程序，可以将在线的数据和设备上的硬件特征结合起来，来提供更加丰富的用户体验。

（4）GPRS、EDGE、3G

现代的移动设备提供了很多种访问 Internet 的途径。大概地看一下，Android 提供了 GPRS、EDGE、3G 3 种连接网络的技术。每一种都清晰地提供应用层。

通过提供移动数据计划的服务商来访问移动网络。

（5）Wi-Fi

Wi-Fi 无线接收和移动热点正在变得越来越常见。

8.5.1 连接Internet资源

在能访问 Internet 资源之前，需要首先在应用程序 manifest 中添加一个 Internet 的 uses-permission 节点，如下面的 XML 片段所示。

```
s-permission android:name=”android.permission.INTERNET”/>
```

接下来的代码给出了打开一个 Internet 数据流的基本样式。

```
String myFeed = getString(R.string.my_feed);
try {
URL url = new URL(myFeed);
URLConnection connection = url.openConnection();
HttpURLConnection httpConnection = (HttpURLConnection)connection;
int responseCode = httpConnection.getResponseCode();
if (responseCode == HttpURLConnection.HTTP_OK) {
InputStream in = httpConnection.getInputStream();
[ ... Process the input stream as required ... ]
}
}
catch (MalformedURLException e) { }
catch (IOException e) { }
```

Android 包含了一些类，帮助用户处理网络通信。它们放在 java.net.*包和 android.net.*包中。

8.5.2 利用Internet资源

Android 提供了很多种利用 Internet 资源的方式。例如，可以在 Activity 中使用包含一个基于 WebKit 的浏览器的 WebView Widget。另一个极端，可以使用客户端的 API，例如 GData API 来直接处理。介于两者之间，可以使用基于 Java 的 XML 解析工具（如 SAX 或 javax）处理远程 XML 种子来提取和访问数据。



解析 XML 和对特定 Web 服务处理的指令阐述并不在本书的范围之内。那就是说，在本章后面的地震例子，给出了一个使用 javax 类来解析 XML 种子的完整的可工作的代码。

如果在程序中使用 Internet 资源，记住一点，用户的数据连接依赖于他们能获得的通信技术。EDGE 和 GSM 是出了名的低带宽，而 Wi-Fi 连接在手机设定中可能是不可靠的。

通过限制传输数据的量来优化用户体验，并且确保应用程序足够强壮来应付网络的掉线和带宽限制。

注意： 在本书后面内容中，将专门通过一章的内容，来举例说明 Android 在 Internet 方面的应用。

第 9 章　Android的数据存储操作

在前面的章节中，已经多次涉及了 Android 的数据存储知识。数据存储是手机领域中的最常见应用之一，通过数据存储能够在移动设备中显示不同的信息。本章详细讲解 Android 数据存储的基本知识，并通过具体实例的实现过程来讲解其使用流程。

9.1　什么是Android数据存储

典型的桌面操作系统提供一种公共文件系统，即任何应用软件可以使用它来存储和读取文件，该文件也可以被其他的应用软件所读取（会有一些权限控制设定）。Android 采用了一种不同的系统，在 Android 中，所有的应用软件数据（包括文件）为该应用软件所私有。然而，Android 同样也提供了一种标准方式供应用软件将私有数据开放给其他应用软件。在本章的内容中，将会描述一个应用软件存储和获取数据、开放数据给其他应用软件、从其他应用软件请求数据并开放它们的多种方式。

在 Android 中，可供选择的存储方式有如下 5 种。

- ❑ SharedPreferences。
- ❑ 文件存储。
- ❑ SQLite 数据库方式。
- ❑ 内容提供器（Content provider）。
- ❑ 网络。

在接下来的内容中，将详细介绍上述存储方式的基本知识。

9.2　SharedPreferences存储

SharedPreferences 是 Android 提供用来存储一些简单的配置信息的一种机制。例如，一些默认欢迎语、登录的用户名和密码等。SharedPreferences 是以键值对的方式存储，这样开发人员可以很方便地实现读取和存入。

9.2.1　SharedPreferences存储类效率分析

SharedPreferences 是 Android 平台上一个轻量级的存储类，主要是保存一些常用的配置比如窗口状态，在 Activity 中重载窗口状态 onSaveInstanceState 一般使用 SharedPreferences 完成，它提供了 Android 平台常规的 Long 长整形、Int 整形、String 字符串型的保存，它是什么样的处理方式呢？

SharedPreferences 类似过去 Windows 系统上的 ini 配置文件，但是它分为多种权限，可以



全局共享访问，Android123 提示最终是以 XML 方式来保存，整体效率来看不是特别的高，对于常规的轻量级而言比 SQLite 要好不少，如果真的存储量不大可以考虑自己定义文件格式。XML 处理时，Dalvik 会通过自带底层的本地 XML Parser 解析，比如 XMLpull 方式，这样对于内存资源占用比较好。

2 个 activity 之间的数据传递除了可以通过 Intent 来传递，还可以使用 SharedPreferences 来共享数据的方式。SharedPreferences 用法很简单，如在 A 中设置如下代码。

```
Editor sharedata = getSharedPreferences("data", 0).edit();
    sharedata.putString("item","hello getSharedPreferences");
    sharedata.commit();
```

然后即可在 B 中编写获取代码。

```
SharedPreferences sharedata = getSharedPreferences("data", 0);
String data = sharedata.getString("item", null);
Log.v("cola","data="+data);
```

通过以下 Java 代码将数据显示出来。

```
<SPAN class=hilite1>SharedPreferences</SPAN> sharedata = getSharedPreferences("data", 0);
String data = sharedata.getString("item", null);
Log.v("cola","data="+data);
```

SharedPreferences 的用法基本上和 J2SE(java.util.prefs.Preferences)中的用法一样，以一种简单、透明的方式来保存一些用户个性化设置的字体、颜色、位置等参数信息。一般的应用程序都会提供“设置”或者“首选项”的这样的界面，那么这些设置最后就可以通过 Preferences 来保存，而程序员不需要知道它到底以什么形式保存的，保存在了什么地方。当然，如果希望保存其他的东西，也没有什么限制。

在 Android 系统中，这些信息以 XML 文件的形式保存在 /data/data/PACKAGE_NAME /shared_prefs 目录下。

（1）数据读取

可以通过下面的代码实现数据读取。

```
String PREFS_NAME = "Note.sample.roiding.com";
SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
boolean silent = settings.getBoolean("silentMode", false);
String hello = settings.getString("hello", "Hi");
```

然后通过以下代码显示出来。

```
String PREFS_NAME = "Note.sample.roiding.com";
<SPAN class=hilite1>SharedPreferences</SPAN>
settings =getSharedPreferences(PREFS_NAME, 0);
boolean silent = settings.getBoolean("silentMode", false);
String hello = settings.getString("hello", "Hi");
```

在上述代码中，“SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);”通过名称，得到一个 SharedPreferences，顾名思义，这个 Preferences 是共享的，共享的范围据现在同一个 Package 中，这里面所说的 Package 和 Java 里面的那个 Package 不同，这里面的 Package 是指在 AndroidManifest.xml 文件中的：

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
package="com.roiding.sample.note"
android:versionCode="1"
android:versionName="1.0.0">
```

对于“boolean silent = settings.getBoolean(”silentMode”, false);”，获得一个 boolean 值，这里就会看到用 Preferences 的好处了：可以提供一个默认值。也就是说如果 Preference 中不存在这个值的话，那么就用后面的值作为返回指，这样就省去了 if 什么什么为空的判断。

（2）数据写入

可以通过的下面的代码实现数据写入。

```
String PREFS_NAME = "Note.sample.roiding.com";
SharedPreferences settings = getSharedPreferences(PREFS_NAME, 0);
SharedPreferences.Editor editor = settings.edit();
editor.putBoolean("silentMode", true);
editor.putString("hello", "Hello~");
editor.commit();
```

注意：访问接口和优先修改数据,并由 getSharedPreferences(String,int)返回数据，为了统一设置参数，有一个单例类供所有的客户端共享。修改参数必须通过一个 SharedPreferences.Editor 对象，在存储它们时，以确保参数值有统一的状态和控制，当前此类不支持多线程。

9.2.2 应用实例

本节通过一个实例的实现过程，来讲解 SharedPreferences 的使用过程。本实例保存在“光盘:\daima\9\SharedPreferencesUsage”，具体实现过程如下。

第 1 步：编写主布局文件 main.xml，具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
```



```
</LinearLayout>
```

第 2 步：编写文件 SharedPreferencesHelper.java，具体代码如下所示。

```
package com.android.SharedPreferences;

import android.content.Context;
import android.content.SharedPreferences;

public class SharedPreferencesHelper {
        SharedPreferences sp;
        SharedPreferences.Editor editor;

        Context context;

        public SharedPreferencesHelper(Context c,String name){
                context = c;
                sp = context.getSharedPreferences(name, 0);
                editor = sp.edit();
        }

        public void putValue(String key, String value){
                editor = sp.edit();
                editor.putString(key, value);
                editor.commit();
        }

        public String getValue(String key){
                return sp.getString(key, null);
        }

}
```

看文件 SharedPreferencesUsage.java，具体代码如下所示。

```
package com.android.SharedPreferences;

import android.app.Activity;
import android.os.Bundle;
import android.widget.TextView;

/*
  * to access from: data/data/com.android.SharedPreferences/share_prefs
  */
public class SharedPreferencesUsage extends Activity {
        public final static String COLUMN_NAME ="name";
        public final static String COLUMN_MOBILE ="mobile";
```

```
        SharedPreferencesHelper sp;
        /** Called when the activity is first created. */
       @Override
       public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            //setContentView(R.layout.main);
            sp = new SharedPreferencesHelper(this, "contacts");
            //1. to store some value
            sp.putValue(COLUMN_NAME, "Gryphone");
            sp.putValue(COLUMN_MOBILE, "123456789");


            //2. to fetch the value
            String name = sp.getValue(COLUMN_NAME);
            String mobile = sp.getValue(COLUMN_MOBILE);

            TextView tv = new TextView(this);
            tv.setText("NAME:"+ name + "\n" + "MOBILE:" + mobile);

            setContentView(tv);
       }
   }
```

至此，整个实例介绍完毕，执行后的效果如图 9-1 所示。

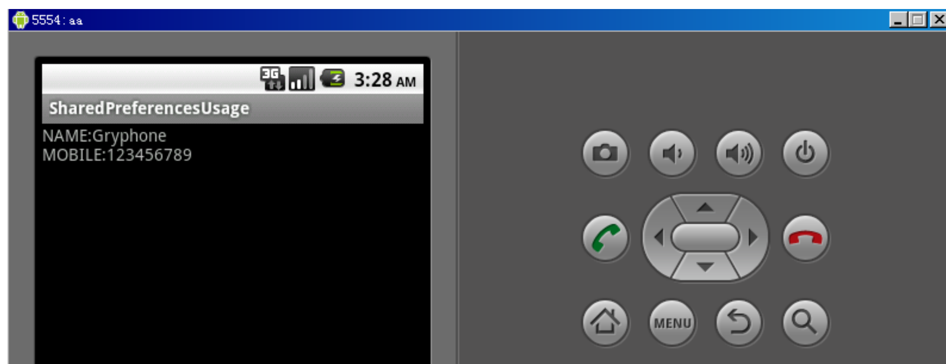


图 9-1　执行效果

这样，“NAME”和“MOBILE”就是在 SharedPreferences 中存储的。

注意：因为上面例子的pack_name为package com.android.SharedPreferences;所以存放数据的路径为：data/data/com.android.SharedPreferences/share_prefs/contacts.xml，如图 9-2 所示。

其中文件 contacts.xml 的内容如下。

```
<?xml version='1.0' encoding='utf-8' standalone='yes' ?>
<map>
<string name="mobile">123456789</string>
```



```
<string name="name">Gryphone</string>
</map>
```

9.3　文件存储

前面介绍的 Shared Preferences 存储方式非常方便，但是其只适合存储比较简单的数据，如果需要存储更多的数据，可行选择的方式有好几种，这里先给读者介绍文件存储的方法。

和传统的 Java 中实现 I/O 的程序类似，在 Android 中，其提供了 openFileInput 和 openFileOuput 方法读取设备上的文件，下面举个例子代码，具体如下所示。

```
String FILE_NAME = "tempfile.tmp";                          //确定要操作文件的文件名
  FileOutputStream fos = openFileOutput(FILE_NAME, Context.MODE_PRIVATE); //初始化
  FileInputStream fis = openFileInput(FILE_NAME);           //创建写入流代码解释
```

在上述代码中，这 2 个方法只支持读取该应用目录下的文件，读取非其自身目录下的文件将会抛出异常。需要提醒的是，如果调用 FileOutputStream 时指定的文件不存在，Android 会自动创建它。另外，在默认情况下，写入的时候会覆盖原文件内容，如果想把新写入的内容附加到原文件内容后，则可以指定其模式为 Context.MODE_APPEND。

通常默认情况下，使用 openFileOutput 方法创建的文件只能被其调用的应用使用，其他应用无法读取这个文件，如果需要在不同的应用中共享数据，可以使用 Content Provider 实现。

注意：关于 Content Provider 将在稍后的内容中介绍。

如果应用需要一些额外的资源文件，例如，一些用来测试用户写的音乐播放器是否可以正常工作的 MP3 文件，可以将这些文件放在应用程序的/res/raw/下，如 mydatafile.mp3。那么就可以在应用中使用 getResources 获取资源后，以 openRawResource 方法（不带后缀的资源文件名）打开这个文件，实现代码如下所示。

```
Resources myResources = getResources();
   InputStream myFile = myResources.openRawResource(R.raw.myfilename);
```

除了前面介绍的读写文件外，Android 还提供了诸如 deleteFile、fileList 等方法来操作文件，在此将不再赘述。

9.4　再看SQLite存储方式

本节开始学习 Android 应用的另外一个方面：将数据存储到文件系统或者数据库当中。当然最经常的是，用户将自己的数据存储到 SQLite 数据库当中。SQLite 是 Android 所带的一个标准的数据库，它支持 SQL 语句，它是一个轻量级的嵌入式数据库。

在下面的内容中，将通过一个实例来演示 SQLite 的使用流程。本实例保存在“光盘:\daima\9\SQLite”，具体演示过程如下。

第 1 步：打开 Eclipse，打开 SQLite 项目。

第 2 步：单击运行项目，此时可以看到主界面如图 9-2 所示，这个界面的布局信息都在 main.xml 文件中，在 1 个 LinearLayout 当中数值排列了 5 个 Button。

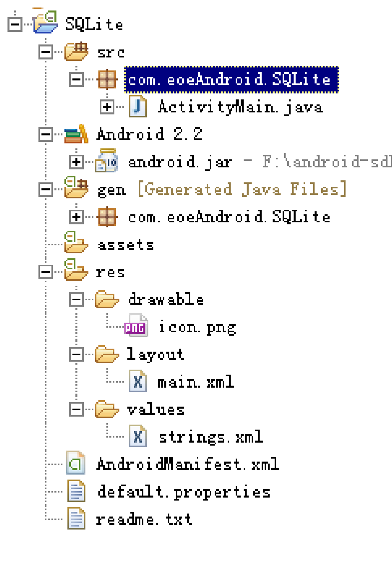


图 9-2　主界面

第 3 步：查看主文件 ActivityMain，具体流程如下。

1）定义继承于 SQLiteOpenHelper 的类 DatabaseHelper，具体代码如下所示。

```
private static class DatabaseHelper extends SQLiteOpenHelper {
    DatabaseHelper(Context context) {
        super(context, DATABASE_NAME, null, DATABASE_VERSION);
    }

    @Override
    public void onCreate(SQLiteDatabase db) {

        String sql = "CREATE TABLE " + TABLE_NAME + " (" + TITLE
                + " text not null, " + BODY + " text not null " + ");";
        Log.i("haiyang:createDB=", sql);
        db.execSQL(sql);
    }
    @Override
    public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
    }
}
```

在上述代码中，DatabaseHelper 类继承了 SQLiteOpenHelper 类，并且重写了 onCreate 和 onUpgrade 方法。在 onCreate()方法里面首先构造一条 SQL 语句，然后调用 db.execSQL(sql)执行 SQL 语句。这条 SQL 语句为用户生成了一张数据库表。

目前还不需要升级数据库，所以在 onUpgrade()函数里边没有执行任何操作。



SQLiteOpenHelper 是一个辅助类，此类主要用于生成一个数据库，并对数据库的版本进行管理。当在程序当中调用这个类的方法 getWritableDatabase()或者 getReadableDatabase()方法的时候，如果当时没有数据，那么 Android 系统就会自动生成一个数据库。SQLiteOpenHelper 是一个抽象类，通常需要继承它，并且实现里边的 3 个函数，各个函数的具体说明如下。

- onCreate（SQLiteDatabase）：在数据库第一次生成的时候会调用这个方法，一般在这个方法里边生成数据库表。
- onUpgrade（SQLiteDatabase, int, int）：当数据库需要升级的时候，Android 系统会主动调用这个方法。一般在这个方法里边删除数据表，并建立新的数据表，当然是否还需要做其他的操作，完全取决于应用的需求。
- onOpen（SQLiteDatabase）：这是当打开数据库时的回调函数，一般也不会用到。

2）编写按钮处理事件：如果单击插入两条记录的按钮，如果数据成功插入到数据库当中的 diary 表中，那么在界面的 title 区域就会有成功的提示，如图 9-3 所示。

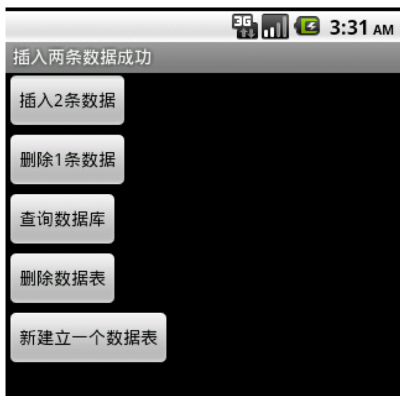


图 9-3　插入成功

单击这个按钮后，程序执行了监听器里的 onClick 方法，并最终执行了上述程序里的 insertItem 方法，其具体代码如下所示。

```
 /* 插入两条数据 */
private void insertItem() {
/* 得到一个可写的 SQLite 数据库，如果这个数据库还没有建立 */
/* 那么 mOpenHelper 辅助类负责建立这个数据库 */
/* 如果数据库已经建立，那么直接返回一个可写的数据库 */
SQLiteDatabase db = mOpenHelper.getWritableDatabase();
String sql1 = "insert into " + TABLE_NAME + " (" + TITLE + ", " + BODY
            + ") values('haiyang', 'android 的发展真是迅速啊');";
String sql2 = "insert into " + TABLE_NAME + " (" + TITLE + ", " + BODY
            + ") values('icesky', 'android 的发展真是迅速啊');";
try {
      Log.i("haiyang:sql1=", sql1);
      Log.i("haiyang:sql2=", sql2);
      db.execSQL(sql1);
      db.execSQL(sql2);
```

```
                setTitle("插入两条数据成功");
            } catch (SQLException e) {
                setTitle("插入两条数据失败");
            }
        }
```

关于对上述代码的一点说明如下。

- sql1 和 sql2：是构造的标准的插入 SQL 语句，如果对 SQL 语句不是很熟悉，可以参考相关的书籍。鉴于本书的重点是在 Android 方面，所以对 SQL 语句的构建不进行详细的介绍。
- Log.i（）：会将参数内容打印到日志当中，并且打印级别是 Info 级别，在使用 LogCat 工具的时候会进行详细的介绍。
- db.execSQL（sql1）：执行 SQL 语句。

注意： Android 支持 5 种打印输出级别，分别是 Verbose、Debug、Info、Warning、Error，当然在程序当中一般用到的是 Info 级别，即将一些自己需要知道的信息打印出来，如图 9-4 所示。

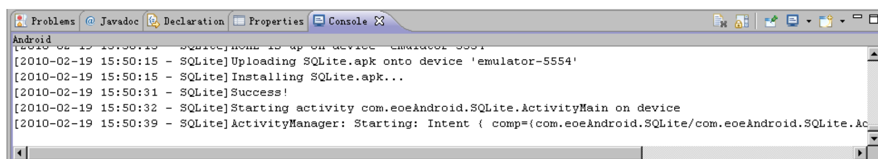


图 9-4　打印输出级别

3）单击查询数据库的按钮，会在界面的 title 区域显示当前数据表当中数据的条数，刚才插入了两条，那么现在单击后应该显示为两条，如图 9-5 所示。

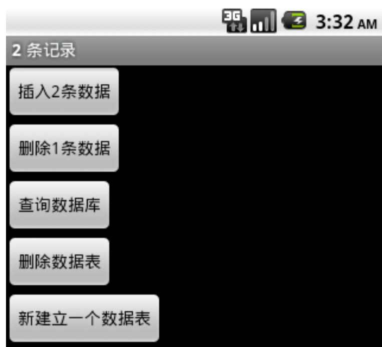


图 9-5　查询数据

单击这个按钮后，程序执行了监听器里的 onClick 方法，并最终执行了上述程序里的 showItems 方法，具体代码如下所示。

```
    /* 在屏幕的 title 区域显示当前数据表当中的数据的条数 */
    private void showItems() {
        /* 得到一个可写的数据库 */
        SQLiteDatabase db = mOpenHelper.getReadableDatabase();
```



```
                String col[] = { TITLE, BODY };
                Cursor cur = db.query(TABLE_NAME, col, null, null, null, null, null);
                /*  通过 getCount()方法，可以得到 Cursor 当中数据的个数  */
                Integer num = cur.getCount();
                setTitle(Integer.toString(num) + "  条记录");
            }
        }
```

在上述代码中，语句“Cursor cur = db.query(TABLE_NAME, col, null, null, null, null, null)”比较难以理解，此语句用于查询到的数据放到一个 Cursor 当中。这个 Cursor 里边封装了这个数据表 TABLE_NAME 当中的所有条列。query()方法相当有用，包含了 7 个参数，各个参数的具体说明如下。

- 第 1 个参数是数据库里边表的名字，比如在这个例子中，表的名字就是 TABLE_NAME，也就是“diary”。
- 第 2 个字段是想要返回数据包含的列的信息。在这个例子当中想要得到的列有 title、body。把这两个列的名字放到字符串数组里边来。
- 第 3 个参数为 selection，相当于 SQL 语句的 where 部分，如果想返回所有的数据，那么就直接置为 null。
- 第 4 个参数为 selectionArgs。在 selection 部分，有可能用到“?”，那么在 selectionArgs 定义的字符串会代替 selection 中的“?”。
- 第 5 个参数为 groupBy。定义查询出来的数据是否分组，如果为 null 则说明不用分组。
- 第 6 个参数为 having ，相当于 SQL 语句当中的 having 部分。
- 第 7 个参数为 orderBy，来描述期望的返回值是否需要排序，如果设置为 null 则说明不需要排序。

注意：Cursor 在 Android 当中是一个非常有用的接口，通过 Cursor 可以对从数据库查询出来的结果集进行随机的读写访问。

4）单击删除一条数据库的按钮后，如果成功删除，可以看到在屏幕的标题（title）区域有文字提示，如图 9-6 所示。

现在再单击查询数据库按钮，看数据库里边的记录是不是少了一条。单击查询数据库按钮后，出现如图 9-7 所示的界面。



图 9-6　删除一条数据

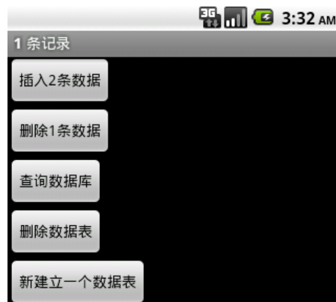


图 9-7　查询数据



当单击删除一条记录的按钮后，程序执行了监听器里的 onClick 方法，并最终执行了上述程序里的 deleteItem 方法，其代码如下所示。

```
/* 删除其中的一条数据 */
private void deleteItem() {
    try {
        SQLiteDatabase db = mOpenHelper.getWritableDatabase();
        db.delete(TABLE_NAME, " title = 'haiyang'", null);
        setTitle("删除 title 为 haiyang 的一条记录");
    } catch (SQLException e) {
    }
}
```

在上述代码中，通过“db.delete(TABLE_NAME, " title = 'haiyang'", null)”语句删除了一条 title='haiyang'的数据。当然如果有很多条数据 title 都为'haiyang'，那么一并删除。delete 方法各个参数的具体说明如下。

- 第一个参数是数据库表名，在这里是 TABLE_NAME，也就是 diary。
- 第二个参数，相当于 SQL 语句当中的 where 部分，也就是描述了删除的条件。

如果在第二个参数当中有“？”符号，那么第三个参数中的字符串会依次替换在第二个参数当中出现的“？”符号。

5）单击删除数据表，可以删除 diary 这张数据表，如图 9-8 所示。



图 9-8　删除表

下边开始看代码的实现部分，具体代码如下所示。

```
/* 删除数据表 */
private void dropTable() {
    SQLiteDatabase db = mOpenHelper.getWritableDatabase();
    String sql = "drop table " + TABLE_NAME;
    try {
        db.execSQL(sql);
        setTitle("数据表成功删除：" + sql);
    } catch (SQLException e) {
        setTitle("数据表删除错误");
    }
}
```



在上述代码中，构造了一个标准的删除数据表的 SQL 语句，然后执行这条语句 db.execSQL(sql)。

6）单击其他的按钮，程序运行时有可能会出现异常，在此单击重新建立数据表按钮，如图 9-9 所示。

现在单击查询数据库，看里边是否有数据，如图 9-10 所示。

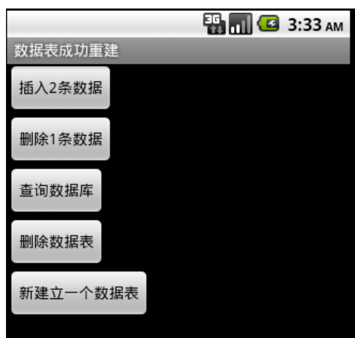


图 9-9　新建表



图 9-10　显示 0 条记录

下边看一下程序是如何建立一张新表的，具体实现代码如下所示。

```
    /* 重新建立数据表 */
    private void CreateTable() {
        SQLiteDatabase db = mOpenHelper.getWritableDatabase();
        String sql = "CREATE TABLE " + TABLE_NAME + " (" + TITLE
                + " text not null, " + BODY + " text not null " + ");";
        Log.i("haiyang:createDB=", sql);
        try {
            db.execSQL("DROP TABLE IF EXISTS diary");
            db.execSQL(sql);
            setTitle("数据表成功重建");
        } catch (SQLException e) {
            setTitle("数据表重建错误");
        }
    }
```

在上述代码中，sql 变量表示的语句为标准的 SQL 语句，负责按要求建立一张新表；“db.execSQL("DROP TABLE IF EXISTS diary")”语句表示，如果存在 diary 表，则需要先删除，因为在同一个数据库当中不能出现两张同样名字的表；“db.execSQL(sql)”语句用于执行 SQL 语句，建立一个新表。

9.5　ContentProvider存储

在第 8 章中，曾经介绍了组成 Android 程序的主要 4 部分，如下所示。

- ❑ Activity。

- Broadcast Intent Receiver。
- Service。
- Content Provider。

本节将详细介绍 Android 中另外一个非常重要的部分 ContentProvider。

9.5.1 Content Provider基础

在 Android 系统中，数据是私有的，当然这些数据包括文件数据和数据库数据以及一些其他类型的数据。Android 中的两个程序之间可以进行数据交换，此功能就是通过 ContentProvider 实现的。一个 Content Provider 类实现了一组标准的方法接口，从而能够让其他的应用保存或读取此 Content Provider 的各种数据类型。也就是说，一个程序可以通过实现一个 Content Provider 的抽象接口将自己的数据暴露出去。外界根本看不到，也不用看到这个暴露的数据在应用当中是如何存储的，或者是用数据库存储还是用文件存储，还是通过网上获得，这些一切都不重要，重要的是外界可以通过这一套标准及统一的接口和程序里的数据打交道，可以读取程序的数据，也可以删除程序的数据，当然，中间也会涉及一些权限的问题。现实中比较常见的接口如下。

- Query（URI URI, String[] projection, String selection, String[] selectionArgs,String sortOrder）：通过 URI 进行查询，返回一个 Cursor。
- insert（URI URL, ContentValues values）：将一组数据插入到 URI 指定的地方。
- update（URI URI, ContentValues values, String where, String[] selectionArgs）：更新 URI 指定位置的数据。
- delete（URI URL, String where, String[] selectionArgs）：删除指定 URI 并且符合一定条件的数据。

1．ContentResolver接口

外界的程序通过 ContentResolver 接口可以访问 ContentProvider 提供的数据，在 Activity 当中通过 getContentResolver()可以得到当前应用的 ContentResolver 实例。ContentResolver 提供的接口和 ContentProvider 中需要实现的接口对应，具体来说主要有以下几个。

- query（URI URI, String[] projection, String selection, String[] selectionArgs,String sortOrder）：通过 URI 进行查询，返回一个 Cursor。
- insert（URI URL, ContentValues values）：将一组数据插入到 URI 指定的地方。
- update（URI URI, ContentValues values, String where, String[] selectionArgs）：更新 URI 指定位置的数据。
- delete（URI URL, String where, String[] selectionArgs）：删除指定 URI 并且符合一定条件的数据。

2．ContentProvider和ContentResolver中的URI

在 ContentProvider 和 ContentResolver，使用的 URI 的形式通常有 2 种，一种是指定全部数据；另一种是指定某个 ID 的数据，看下面的例子。

```
content://contacts/people/        //此 URI 指定的就是全部的联系人数据
content://contacts/people/1       //此 URI 指定的是 ID 为 1 的联系人的数据
```

在上边两个类中用到的 URI 一般由 3 部分组成，具体说明如下。



- 第一部分是：“content://”。
- 第二部分是要获得数据的一个字符串片段。
- 第三部分是 ID（如果没有指定 ID，那么表示返回全部）。

因为 URI 通常比较长，而且有时候容易出错，且难以理解。所以，在 Android 当中定义了一些辅助类，并且定义了一些常量来代替这些长字符串的使用，例如下边的代码。

```
Contacts.People.CONTENT_URI （联系人的 URI）
```

9.5.2 使用ContentProvider

下面将通过一个实例来演示 ContentProvider 的使用流程。本实例保存在“光盘:\daima\9\ContentProvider”，具体过程如下。

第 1 步：打开 Eclipse，打开 ContentProvider 项目，程序的目录结构如图 9-11 所示。

第 2 步：运行此项目，将会看到如图 9-12 所示的界面。

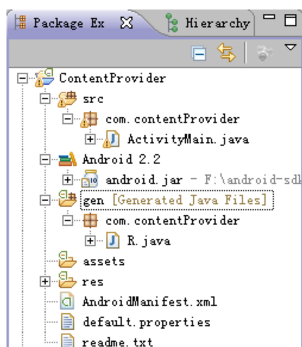


图 9-11　程序目录结构

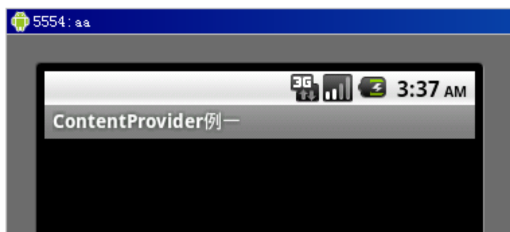


图 9-12　初始效果

第 3 步：开始添加几条数据到联系人列表中，具体流程如下。

1）单击模拟器的键，在弹出的界面上，单击桌面上的“Add”选项，如图 9-13 所示。

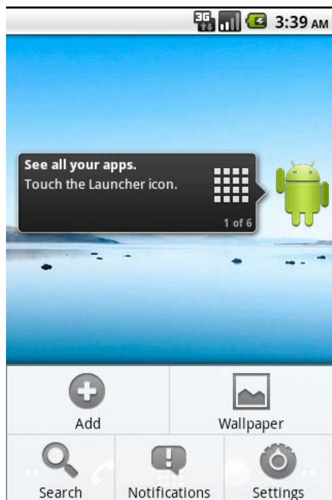


图 9-13　出现的桌面

2）进入应用后，依次单击选择“Shortcuts”、“Contact”和“Create new contact”，如图 9-14 所示。

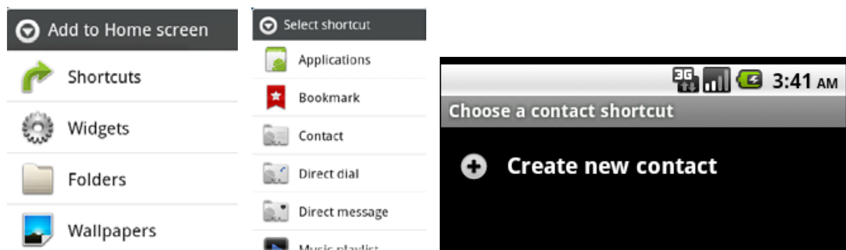


图 9-14　单击 Creat new contact

3）在弹出的界面中添加联系人姓名和电话号码信息，如图 9-15 所示。

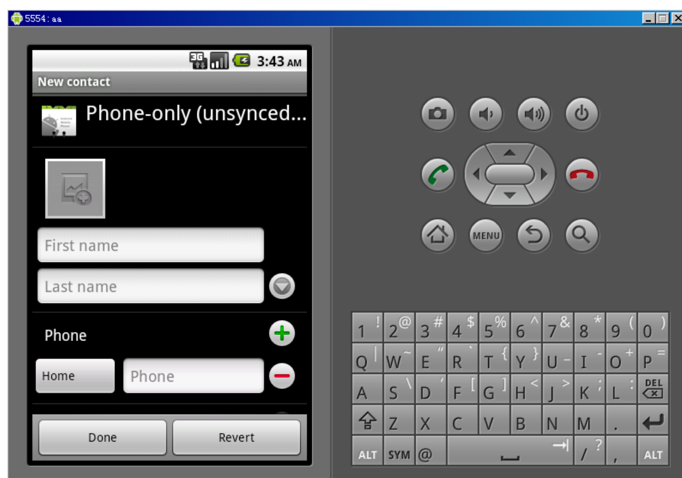


图 9-15　添加联系人姓名和电话号码

4）填写资料完毕后，单击“Done”按钮完成保存，如图 9-16 所示。

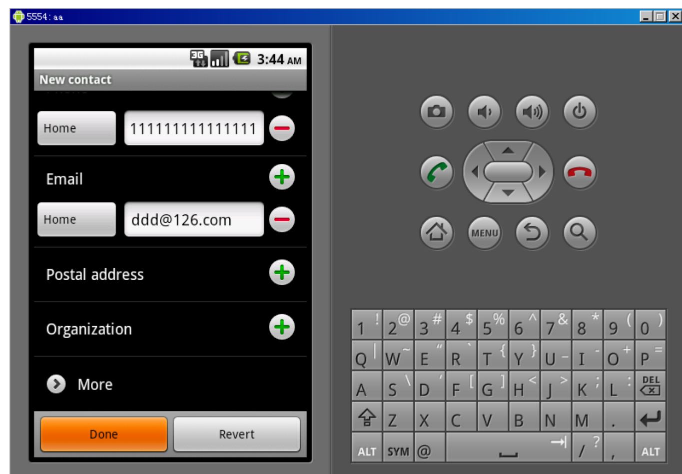


图 9-16　单击 Done 保存



5）按照上述操作步骤，即可添加 1 条联系人数据，效果如图 9-17 所示。

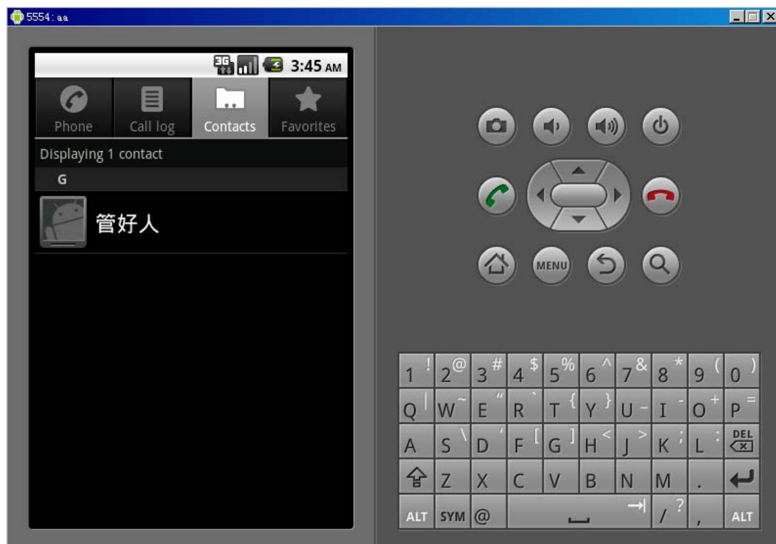


图 9-17　添加后的数据

下面分析程序 ActivityMain 中的 onCreate()方法，其具体代码如下所示。

```
    protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        Cursor c = getContentResolver().query(Phones.CONTENT_URI, null, null, null, null);
        startManagingCursor(c);
        ListAdapter adapter = new SimpleCursorAdapter(this,
                android.R.layout.simple_list_item_2, c,
                        new String[] { Phones.NAME, Phones.NUMBER },
                        new int[] { android.R.id.text1, android.R.id.text2 });
        setListAdapter(adapter);
    }
```

关于对上述代码的解释如下。

1）getContentResolver()方法：得到应用的 ContentResolver 实例。

2）query（Phones.CONTENT_URI, null, null, null, null）：是 ContentResolver 里的方法，负责查询所有联系人，并返回一个 Cursor。这个方法参数比较多，各个参数的具体含义如下。

- 第一个参数为 URI，在这个例子里边这个 URI 是联系人的 URI。
- 第二个参数是一个字符串的数组，数组里边的每一个字符串都是数据表中某一列的名字，它指定返回数据表中哪些列的值。
- 第三个参数相当于 SQL 语句的 where 部分，描述哪些值是用户需要的。
- 第四个参数是一个字符串数组，它里边的值依次代替在第三个参数中出现的“?”符号。
- 第五个参数指定了排序的方式。

3）startManagingCursor(c)语句：让系统来管理生成的 Cursor。

4）ListAdapter adapter = new SimpleCursorAdapter(this,Android.R.layout.simple_list_item_2,

c, new String[] { Phones.NAME, Phones.NUMBER }, new int[] { Android.R.id.text1, Android.R.id.text2 })语句：用于生成一个 SimpleCursorAdapter。

5）setListAdapter(adapter)：将 ListView 和 SimpleCursorAdapter 进行绑定。

9.6 网络存储

本章中前面介绍的几种存储都是将数据存储在本地设备上，除此之外，还有一种存储（获取）数据的方式，通过网络来实现数据的存储和获取，下面看一个在 Android 上调用 WebService 的例子。

在 Android 的早期版本中，曾经支持过进行 XMPP Service 和 Web Service 的远程访问。Android SDK 1.0 以后的版本对它以前的 API 作了许多的变更。Android 1.0 以上版本不再支持 XMPP Service，而且访问 Web Service 的 API 全部变更。

1. 实例介绍

本实例的功能是通过邮政编码查询该地区的天气预报，以 POST 发送的方式发送请求到 webservicex.net 站点，访问 WebService.webservicex.net 站点上提供查询天气预报的服务，具体信息请参考其 WSDL 文档，网址如下。

```
http://www.webservicex.net/WeatherForecast.asmx?WSDL
```

输入：美国某个城市的邮政编码。

输出：该邮政编码对应城市的天气预报。

2. 实现过程

具体实现过程如下。

1）如果需要访问外部网络，则需要在 AndroidManifest.xml 文件中加入如下代码申请权限许可。

```
<!-- Permissions -->
  <uses-permission Android:name="Android.permission.INTERNET" />
```

2）以 HTTP POST 的方式发送，SERVER_URL 并不是指 WSDL 的 URL，而是服务本身的 URL。具体实现的代码如下所示。

```
private static final String SERVER_URL = "http://www.webservicex.net/WeatherForecast. asmx/
GetWeatherByZipCode";                                   //定义需要获取的内容来源地址
  HttpPost request = new HttpPost(SERVER_URL);          //根据内容来源地址创建一个 Http 请求
  // 添加一个变量
  List <NameValuePair> params = new ArrayList <NameValuePair>();
  // 设置一个华盛顿区号
  params.add(new BasicNameValuePair("ZipCode", "200120"));              //添加必须的参数
  request.setEntity(new UrlEncodedFormEntity(params, HTTP.UTF_8));      //设置参数的编码
  try { HttpResponse httpResponse = new DefaultHttpClient().execute(request);  //发送请求并获取反馈
  // 解析返回的内容
  if(httpResponse.getStatusLine().getStatusCode() != 404)
```



```
{
    String result = EntityUtils.toString(httpResponse.getEntity());
    Log.d(LOG_TAG, result);
}
} catch (Exception e) {
Log.e(LOG_TAG, e.getMessage());
}
```

通过上述代码，使用 Http 从 webservicex 获取 ZipCode 为“200120”（美国 WASHINGTON D.C）的内容，其返回的内容如下。


```
<WeatherForecasts xmlns:xsd="http://www.w3.org/2001/XMLSchema" xmlns:xsi="http: //www.w3.org/
2001/XMLSchema-instance" xmlns="http://www.webservicex.net">
    <Latitude>38.97571</Latitude>
    <Longitude>77.02825</Longitude>
    <AllocationFactor>0.024849</AllocationFactor>
    <FipsCode>11</FipsCode>
    <PlaceName>WASHINGTON</PlaceName>
    <StateCode>DC</StateCode>
    <Details>
      <WeatherData>
        <Day>Saturday, April 25, 2009</Day>
       <WeatherImage>http://forecast.weather.gov/images/wtf/sct.jpg</WeatherImage>
        <MaxTemperatureF>88</MaxTemperatureF>
        <MinTemperatureF>57</MinTemperatureF>
        <MaxTemperatureC>31</MaxTemperatureC>
        <MinTemperatureC>14</MinTemperatureC>
      </WeatherData>
      <WeatherData>
        <Day>Sunday, April 26, 2009</Day>
       <WeatherImage>http://forecast.weather.gov/images/wtf/few.jpg</WeatherImage>
        <MaxTemperatureF>89</MaxTemperatureF>
        <MinTemperatureF>60</MinTemperatureF>
        <MaxTemperatureC>32</MaxTemperatureC>
        <MinTemperatureC>16</MinTemperatureC>
      </WeatherData>
…
    </Details>
</WeatherForecasts>
```


通过上述实例，演示了如何在 Android 中通过网络获取数据。要掌握该类内容，开发者需要熟悉 java.net.*，Android.net.*这两个包的内容，具体请读者参阅相关文档。

9.7　数据存储演练

通过前面内容的学习，基本掌握了各种 Android 数据存储的基本知识。为了使读者加深

对前面知识的学习，在本节的内容中，将通过几个实例的实现过程，来讲解 Android 数据存储的具体实现过程。

9.7.1 SQLite实现一个日记本

在前面的内容中，已经对 Android 系统自带的 SQLite 数据库进行了初步的学习，了解了一些增、删、改、查的基本工作。在下面的例子当中，做了一个非常简便的日记本程序。

1. 功能介绍

在本实例中，不但要对数据库进行增、删、改、查的操作，而且还要把数据库当中的数据显示在一个 ListView 当中，通过对 ListView 的操作，实现对数据的增、删、改、查操作。

通过本实例可以实现以下操作。

- 对 DatabaseHelper 和 SQLiteDatabase 封装，以便让用户访问数据库更加方便和安全。
- 利用 ContentValues 类来代替原始的 SQL 语句进行数据库的操作。
- 使用 SimpleCursorAdapter 类和 ListView 配合进行 ListView 的显示。

本实例保存在“光盘\daima\9\ShiYongSQLite”中。

2. 实现过程

本实例的具体实现过程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“ShiYongSQLite”的工程文件。

第 2 步：编写 diary_list.xml 主布局文件，此文件是项目执行后首先显示的界面，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">

    <ListView android:id="@+id/android:list"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView android:id="@+id/android:empty"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="点击下边的 Menu 按钮开始写日记:)" />
</LinearLayout>
```

执行后效果如图 9-18 所示。

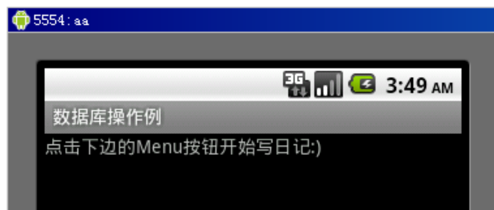


图 9-18 首先显示的界面



第 3 步：使用 DiaryDbAdapter 类，此类用于封装 DatabaseHelper 和 SQLiteDatabase 类，使用户对数据库的操作更加安全和方便。在 DiaryDbAdapter 类变量中，主要定义了以下几个变量。

- 数据库、数据表、数据表中列的名字。
- DatabaseHelper 和 SQLiteDatabase 的实例。
- Context 实例。

定义 DatabaseHelper 类的方法和前面的例子一样，只不过这个例子里边，在 onUpgrade 增加了升级的代码，具体如下所示。

```
private static class DatabaseHelper extends SQLiteOpenHelper {
        DatabaseHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }
        @Override
        public void onCreate(SQLiteDatabase db) {                  //生成数据库
            db.execSQL(DATABASE_CREATE);
        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
            db.execSQL("DROP TABLE IF EXISTS diary");
            onCreate(db);
        }
}
```

在 DiaryDbAdapter 类里，向外界提供了以下一些常用的方法。

- open()：调用这个方法后，如果数据库还没有建立，那么会建立数据库，如果数据库已经建立了，那么会返回可写的数据库实例。
- close()：调用此方法，DatabaseHelper 会关闭对数据库的访问。
- createDiary（String title, String body）通过一个 title 和 body 字段在数据库当中创建一条新的纪录。
- deleteDiary（long rowId）：通过记录的 ID，删除数据库中的那条记录。
- getAllNotes()：得到 diary 表中所有的记录，并且以一个 Cursor 的形式进行返回。
- getDiary（long rowId）：通过记录的主键 ID，得到特定的一条记录。
- updateDiary（long rowId, String title, String body）：更新主键 ID 为 rowId 那条记录中的两个字段 title 和 body 字段的内容。

注意：ContentValues 类和 Hashtable 比较类似，其功能是负责存储一些名值对，但是它存储的名值对当中的名是一个 String 类型，而值都是基本类型。

第 4 步：实现插入日记功能。在前面的实例中，是通过 SQL 语句实现插入操作的，SQL 语句的好处是比较直观，但是容易出错。在这个例子当中，将要插入的值都放到一个 ContentValues 的实例当中，然后执行插入操作，具体代码如下所示。

```
public long createDiary(String title, String body) {
```

```
        ContentValues initialValues = new ContentValues();
        initialValues.put(KEY_TITLE, title);
        initialValues.put(KEY_BODY, body);
        Calendar calendar = Calendar.getInstance();
        String created = calendar.get(Calendar.YEAR) + "年"
                + calendar.get(Calendar.MONTH) + "月"
                + calendar.get(Calendar.DAY_OF_MONTH) + "日"
                + calendar.get(Calendar.HOUR_OF_DAY) + "小时"
                + calendar.get(Calendar.MINUTE) + "分种";
        initialValues.put(KEY_CREATED, created);
        return mDb.insert(DATABASE_TABLE, null, initialValues);
    }
```

在上述代码中，“ContentValues initialValues = new ContentValues()”语句用于实例化一个 contentValues 类；“initialValues.put（KEY_TITLE, title）”语句用于将列名和对应的列值放置到 initialValues 里边；“mDb.insert（DATABASE_TABLE, null, initialValues）”语句负责插入一条新的纪录，如果插入成功则会返回这条记录的 ID，如果插入失败会返回-1。

第 5 步：实现修改日记功能。在更新一条记录的时候，也是采用 ContentValues 的这套机制，具体代码如下所示。

```
    public boolean updateDiary(long rowId, String title, String body) {
        ContentValues args = new ContentValues();
        args.put(KEY_TITLE, title);
        args.put(KEY_BODY, body);
        Calendar calendar = Calendar.getInstance();
        String created = calendar.get(Calendar.YEAR) + "年"
                + calendar.get(Calendar.MONTH) + "月"
                + calendar.get(Calendar.DAY_OF_MONTH) + "日"
                + calendar.get(Calendar.HOUR_OF_DAY) + "小时"
                + calendar.get(Calendar.MINUTE) + "分种";
        args.put(KEY_CREATED, created);

        return mDb.update(DATABASE_TABLE, args, KEY_ROWID + "=" + rowId, null) > 0;
    }
```

通过上述代码，实现了更新一条记录的功能。

第 6 步：返回到程序的主界面，对应的 Activity 是 ActivityMain。当单击【menu】按钮后会出现如图 9-19 所示界面。

单击【MENU】按钮的处理逻辑全部放在 onMenuItem Selected 函数里，具体代码如下所示。

```
    @Override
    public boolean onMenuItemSelected(int featureId, MenuItem item) {
        switch (item.getItemId()) {
        case INSERT_ID:
            createDiary();
            return true;
```



```
            case DELETE_ID:
                mDbHelper.deleteDiary(getListView().getSelectedItemId());
                renderListView();
                return true;
            }
            return super.onMenuItemSelected(featureId, item);
        }
```



图 9-19　运行界面

在上述代码中，如果单击添加一篇新日记按钮那么会执行到 createDiary()语句；如果单击删除一条记录，会执行“mDbHelper.deleteDiary(getListView().getSelectedItemId())”语句，首先删除当前被选中的某一项所对应的数据库当中的记录；“renderListView()”语句用于重新对界面刷新。

在 createDiary()函数的具体代码如下所示。

```
        private void createDiary() {
          Intent i = new Intent(this, ActivityDiaryEdit.class);
          startActivityForResult(i, ACTIVITY_CREATE);
          }
```

在上述代码中，首先构造了一个 Intent，此 Intent 负责跳转到 ActivityDiaryEdit 里。然后启动这个 Intent，并且需要返回值。

第 7 步：在 ActivityMain 中，有多处地方都用到了 renderListView()函数。在 onCreate 里边用这个函数显示 ListView。当 ListView 需要发生变化后，例如，删除了一条记录或者增加了一条记录的时候，调用这个函数进行刷新 ListView，renderListView()函数的具体代码如下所示。

```
        private void renderListView() {
            mDiaryCursor = mDbHelper.getAllNotes();
            startManagingCursor(mDiaryCursor);
            String[] from = new String[] { DiaryDbAdapter.KEY_TITLE,
                    DiaryDbAdapter.KEY_CREATED };
            int[] to = new int[] { R.id.text1, R.id.created };
            SimpleCursorAdapter notes = new SimpleCursorAdapter(this,
                    R.layout.diary_row, mDiaryCursor, from, to);
            setListAdapter(notes);
        }
```

关于对上述代码的几点说明如下。

1）mDiaryCursor = mDbHelper.getAllNotes()语句：首先获取数据库当中的所有数据，这些数据以 Cursor 的形式存在。

2）startManagingCursor（mDiaryCursor）语句：将生成的 Cursor 交给 Activity 来管理，这样的好处是系统能自动做很多事情，比如当程序暂停的时候，这个系统可以卸载 Cursor 以节省空间，当程序重新启动的时候系统重新查询生成 Cursor。

3）String[] from：定义了 ListView 每一排对应的数据是从数据库中的哪个列表里选取。

4）int[] to：和 SimpleAdapter 类似，里边是一个 View 的数组。这些 View 只能是 TextView 或者 ImageView。这些 View 是以 ID 的形式来表示的，如 Android.R.id.text1。

5）SimpleCursorAdapter notes = new SimpleCursorAdapter(this,R.layout.diary_row, mDiaryCursor, from, to)语句：生成一个 SimpleCursorAdapter ，各个参数的具体说明如下。

- 第一个参数是 Context。
- 第二个参数为 R.layout.diary_row，它关联在 diary_row.xml 文件当中定义的 Layout，这个 Layout 规定 ListView 当中每一项的布局。
- 第三个参数就是 Cursor。
- 第四个参数是数据表当中的列的数组，只有在这里边出现的列名，数据才会对应地填充在 to 里边对应的 TextView 或者 ImageView 当中。
- 第五个参数是在 ListView 里边每一项中需要被数据填充的 TextView 或者 ImageView。

6）setListAdapter（notes）语句：将 SimpleCursorAdapter 和 ListActivity 里边的 ListView 绑定起来，至此在界面当中才会显示出列表来。

注意：SimpleCursorAdapter 和 ArrayAdapter 类似，SimpleCursorAdapter 也是集成 Adapter。ArrayAdapter 负责把一个字符串数组中的数据填充到一个 ListView 当中，而对应的 SimpleCursorAdapter 负责把 Cursor 里边的内容填充到 ListView 当中。通过 SimpleCursorAdapter 可以把数据库当中一列的数据和 ListView 中一排进行对应起来。和前两个 Adapter 类似，要求和数据进行对应的 View 必须是 TextView 或者 ImageView。

第 8 步：实现添加日记功能。当单击添加一条数据的按钮，程序运行界面如图 9-20 所示。



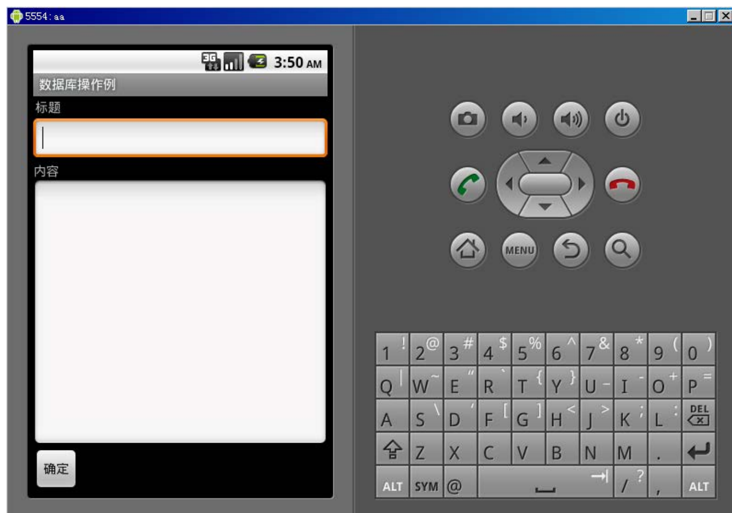


图 9-20　添加日记界面

此界面对应的 Activity 是 ActivityDiaryEdit，对应的布局文件是 diary_edit.xml，对应代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <LinearLayout android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content">
        <TextView android:layout_width="wrap_content"
            android:layout_height="wrap_content" android:text="@string/title"
            android:padding="2px" />
        <EditText android:id="@+id/title"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content" android:layout_weight="1" />
    </LinearLayout>
    <TextView android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="@string/body" />
    <EditText android:id="@+id/body" android:layout_width="fill_parent"
        android:layout_height="fill_parent" android:layout_weight="1"
        android:scrollbars="vertical" android:gravity="top" />
    <Button android:id="@+id/confirm" android:text="@string/confirm"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
</LinearLayout>
```

在上述布局文件中，用到了 LinearLayout 控件、TextView 控件和 EditText 控件，在一个 LinearLayout 里边放置了一些文本框、输入框和一些 Button 按钮。

当程序运行输入内容后，单击确定按钮，日记就会保存到数据库当中。下边来看一下代码具体是怎么执行的。当单击“确定”按钮后，系统回调执行和按钮绑定的单击监听器里边的 onClick 方法，具体代码如下所示。

```
public void onClick(View view) {
   String title = mTitleText.getText().toString();
   String body = mBodyText.getText().toString();
   if (mRowId != null) {
   mDbHelper.updateDiary(mRowId, title, body);
   } else mDbHelper.createDiary(title, body);
   Intent mIntent = new Intent();
   setResult(RESULT_OK, mIntent);
   finish();
   }
```

在上述代码中，首先获得了 EditView 里边的数据。

目前 mRowId 为 null，所以执行 mDbHelper.createDiary（title, body）语句将数据保存到数据当中。

setResult（RESULT_OK, mIntent）语句用于设置返回值。

执行完上边的代码后，系统跳转到 ActivityMain，并执行回调函数 onActivityResult，具体代码如下所示。

```
protected void onActivityResult(int requestCode, int resultCode,
   Intent intent) {
   super.onActivityResult(requestCode, resultCode, intent);
   renderListView();
   }
```

第 9 步：对已经添加的日记进行修改处理。当单击 ListView 里边的条列后，可以对刚才保存的数据进行编辑。上述功能是通过 ActivityMain 中的 onListItemClick 方法实现的，具体代码如下所示。

```
@Override
//  需要对 position 和 id 进行一个很好的区分
// position 指的是点击的这个 ViewItem 在当前 ListView 中的位置
//  每一个和 ViewItem 绑定的数据，肯定都有一个 id，通过这个 id 可以找到那条数据
protected void onListItemClick(ListView l, View v, int position, long id) {
      super.onListItemClick(l, v, position, id);
      Cursor c = mDiaryCursor;
      c.moveToPosition(position);
      Intent i = new Intent(this, ActivityDiaryEdit.class);
      i.putExtra(DiaryDbAdapter.KEY_ROWID, id);
      i.putExtra(DiaryDbAdapter.KEY_TITLE, c.getString(c
                .getColumnIndexOrThrow(DiaryDbAdapter.KEY_TITLE)));
      i.putExtra(DiaryDbAdapter.KEY_BODY, c.getString(c
                .getColumnIndexOrThrow(DiaryDbAdapter.KEY_BODY)));
```



```
            startActivityForResult(i, ACTIVITY_EDIT);
        }
```

关于对上述代码的几点说明。

- c.moveToPosition（position）语句：将在 Cursor 当中的指针移到 position 位置，这个 position 是用户单击的这个一列在整个列表中的位置。
- Intent i = new Intent（this, ActivityDiaryEdit.class）语句：构造一个跳转到 ActivityDiaryEdit 的 Intent。
- putExtra()方法：将要传递的数据放到 Intent 当中。
- c.getString（c.getColumnIndexOrThrow(DiaryDbAdapter.KEY_TITLE)）：得到这一条数据中列名为 title 的值。
- c.getString（c.getColumnIndexOrThrow(DiaryDbAdapter.KEY_BODY)）：得到这一条数据中列名为 body 的值。
- startActivityForResult（i, ACTIVITY_EDIT）语句：启动 Intent，发生 Activity 的跳转。

ActivityDiaryEdit 中的 onCreate()方法的具体代码如下所示。

```
        Bundle extras = getIntent().getExtras();
          if (extras != null) {
          String title = extras.getString(DiaryDbAdapter.KEY_TITLE);
          String body = extras.getString(DiaryDbAdapter.KEY_BODY);
          mRowId = extras.getLong(DiaryDbAdapter.KEY_ROWID);
           if (title != null) {
          mTitleText.setText(title);
          }   if (body != null) {
          mBodyText.setText(body);
          }
          }
```

在上述代码中，ActivityDiaryEdit 中的 Activity 有如下两个 Intent 可以跳转进来。

- 第一种是新建一篇日记的时候，此时 intent 中的 extras 部分没有任何数据。
- 第二种情况是在单击列表的某一个条列的时候，此时的 Intent 如上所示会携带 extras 数据。所以在 ActivityDiaryEdit 中通过判断 extras 是否为 null，就可以判断是哪种 Intent 启动的。

第 10 步：在程序的主界面当中，上下移动焦点（可以通过键盘的上下键或者模拟器中的上下按键），可以对拥有焦点的那一项进行删除，如图 9-21、图 9-22 所示。



图 9-21　上下移动焦点界面



图 9-22　开始删除一条日记

删除操作时，首先执行 onMenuItemSelected 中的代码。

```
mDbHelper.deleteDiary(getListView().getSelectedItemId());
  renderListView();
```

在上述代码中出现如下方法。

- getListView()方法：用于获取当前的 ListView 引用。
- getSelectedItemId()方法用于获取当前这一列所对应的数据项的 rowId，也就是这条数据在数据库中的主键 ID。
- mDbHelper.deleteDiary()方法：用于删去数据库中的这一列数据。
- renderListView()方法：用于对列表重新刷新一遍。

至此，对数据库的学习就先告一段落，接下来将学习 contentProvider，它是 Android 应用当中非常重要的一部分。而且程序间的大部分数据交换都是通过 contentProvider 机制进行。

9.7.2 ContentProvider操作一个数据日记本

在前面的内容中，已经学习了 ContentProvider，并且使用了系统中的一个联系人的 ContentProvider，在本节当中，通过一个实例实现类似上一小节日记本的效果。

1. 功能介绍

在本实例中，日记本功能是通过 ContentProvider 实现的，而不是直接用数据库实现。这样外界的程序就可以访问得到日记本这个应用数据。

通过本实例可以学到以下操作。

- 如何实现一个 ContentProvider。
- 理解 UriMatcher 的含义。
- onPrepareOptionsMenu 方法介绍。

本实例保存在“光盘\daima\9\ShiYongContentProvider”中。

2. 实现过程

本实例的具体实现过程如下。

第 1 步：打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“ShiYongContentProvider”的工程文件。

第 2 步：运行后界面如图 9-23 所示。



图 9-23 执行效果

图 9-23 所示的界面效果在前面的例子里已经见过。本例的主 Activity 仍然是一个 ListActivity，而且关联的布局文件也是 diary_list.xml，其具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
```



```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content">
    <ListView android:id="@+id/android:list"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" />
    <TextView android:id="@+id/android:empty"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content" android:text="点击 Menu 按钮开始写日记:)" />
</LinearLayout>
```

在上述代码中，ListView 的 id 必须定义成 Android:id="@+id/Android:list"的形式，这样系统才可以在 List Activity 里引用的到。

第 3 步：和前面的日记本实例一样，日记本的数据还是存储在 SQLite 数据库中。稍有不同的是，本例中，执行增、删、改、查操作时不是直接访问数据库，而是通过日记本程序的 ContentProivder 来实现。

下面首先看一下 Diary 类，在此类中有一个内部静态类 DiaryColumns，和它的名字意思一样，此类里主要定义了日记本数据库的列表字段的名字，具体代码如下所示。

```
public static final class DiaryColumns implements BaseColumns {
    private DiaryColumns() {}
    public static final Uri CONTENT_URI = Uri.parse("content://" + AUTHORITY + "/diaries");
    public static final String CONTENT_TYPE = "vnd.Android.cursor.dir/vnd.google. diary";
    public static final String CONTENT_ITEM_TYPE = "vnd.Android.cursor.item/vnd. google.diary";
    public static final String DEFAULT_SORT_ORDER = "created DESC";
    public static final String TITLE = "title";
    public static final String BODY = "body";
    public static final String CREATED = "created";
}
```

在上述代码中，BaseColumns 是一个接口，里边有两个变量，一个是_ID="_id"，一个是_COUNT="_ count"。在 Android 中，每一个数据库表至少有一个字段，而且这个字段是_id。所以当用户构造列名的辅助类时，直接实现 BaseColumns，这样便默认地拥有了_id 字段。

在本实例的数据表里，一共有 4 个字段，分别是：“id”、“title”、“body”、“created”。

第 4 步：开始分析 DiaryContentProvider 类，此类继承自 ContentProvider，在此类中实现了 ContentProvider 的一些接口方法，并且成为日记本的一个 ContentProvider。现在通过学习这个类，来详细了解实现一个自定义的 ContentProvider 的具体过程。

1）继承 ContentProvider.DiaryContentProvider 类是继承 ContentProvider 类的。

2）定义一个 public staticfinal 的 URI 类型的变量，并且命名为 CONTENT_URI。Diary Content Provider 所能处理的 URI 都是基于 CONTENT_URI 来构建的，需要注意的是，这个 CONTENT_URI 中的内容以 content://开头，并且全部小写，且全局唯一。

下边是在这个例子中的一个普通 URI，通过分析这个 URI，可以了解 content URI 的构成，代码如下所示。

```
content://com.shiyongcontentprovider.diarycontentprovider/diaries/1
```

代码解释如下。

- 第一部分：是 content://，这部分是固定存在的，不需要做任何修改。
- 第二部分：是授权（AUTHORITY）部分，在本例中是 com.shiyongcontentprovider.diarycontentprovider，授权部分是唯一的，在程序中一般是用户实现的那个 Content Provider 的全称，并且全都小写。这部分是和在 AndroidManifest.xml 文件当中的如下部分是对应的。

```
<provider Android:name ="DiaryContentProvider
Android:authorities="shiyongcontentprovider. diaryconten tprovider" />
```

- 第三部分：是请求数据的类型，例如，在这个例子当中定义的类型是 diaries。当然这一部分可以是 0 个片段或者多个片段构成，如果 Content Provider 只是暴露出了一种类型的数据，那么这部分可以为空，但是如果暴露出了多种，尤其是包含子类的时候，就不能为空，例如，日记本程序里边可以暴露出来两种数据，一种是用户自己的“diaries/my”；另一种是其他人的“diaries/others”。
- 第四部分：是“1”，当然这部分是允许为空的。如果为空，表示请求全部数据；如果不为空，表示请求特定 ID 的数据。

3）构建用户的数据存储系统。在这个例子当中，是将数据存储到数据库系统当中。通常是将数据存储在数据库系统中，但是也可以将数据存储在其他的地方，如文件系统等。

4）实现 ContentProvider 这个抽象类的抽象方法，具体如下所示。

- public boolean onCreate()：当 ContentProvider 生成的时候调用此方法。
- public Cursor query(URI URI, String[] projection, String selection,String[] selectionArgs, String sortOrder)：此方法返回一个 Cursor 对象作为查询结果集。
- public Uri insert（URI URI, ContentValues initialValues）：此方法负责往数据集当中插入一列，并返回这一列的 URI。
- public int delete（URI URI, String where, String[] whereArgs）：此方法负责删除指定 URI 的数据。
- public int update（URI URI, ContentValues values, String where,String[] whereArgs）：此方法负责更新指定 URI 的数据。
- public String getType（URI URI）：返回所给 URI 的 MIME 类型。

5）在 AndroidManifest.xml 文件中增加<provider>标签，例如，在本例中的实现的代码如下。

```
<provider Android:name="DiaryContentProvider" Android:authorities="com.shiyongcontentprovider.
diarycontentprovider" />
```

其中 Android:name 为用户实现的这个 content provider 的类名；Android: authorities 为 content URI 的第二部分，即授权部分，代码为："com.shiyongcontentprovider. diarycontentprovider"。

第 5 步：继续来看 DiaryContentProvider 类的代码。此实例的数据是存储在数据库当中，和前面章节学过的数据例子一样，此实例中定义了 DatabaseHelper 辅助类。这个类在第 8 章



中有详细的讲解，这里就不再进行详细说明，具体代码如下所示。

```
private static class DatabaseHelper extends SQLiteOpenHelper {
        DatabaseHelper(Context context) {
                super(context, DATABASE_NAME, null, DATABASE_VERSION);
                Log.i("jinyan", "DATABASE_VERSION=" + DATABASE_VERSION);
        }
        @Override
        public void onCreate(SQLiteDatabase db) {
                Log.i("jinyan", "onCreate(SQLiteDatabase db)");
                String sql = "CREATE TABLE " + DIARY_TABLE_NAME + " ("
                                + DiaryColumns._ID + " INTEGER PRIMARY KEY,"
                                + DiaryColumns.TITLE + " TEXT," + DiaryColumns.BODY
                                + " TEXT," + DiaryColumns.CREATED + " TEXT" + ");";
                //
                sql ="CREATE TABLE " + DIARY_TABLE_NAME + " ("
                + DiaryColumns._ID + " INTEGER PRIMARY KEY,"
                + DiaryColumns.TITLE + " varchar(255)," + DiaryColumns.BODY
                + " TEXT," + DiaryColumns.CREATED + " TEXT" + ");";
                Log.i("jinyan", "sql="+sql);
                db.execSQL(sql);
        }
        @Override
        public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
                Log.i("jinyan",
                                " onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion)="
                                                + newVersion);
                db.execSQL("DROP TABLE IF EXISTS diary");
                onCreate(db);
        }
}
```

在上述代码中，DatabaseHelper 里操作数据库的辅助类，通过此类可以生成数据库，并且维护这个数据库。

在 DiaryContentProvider 中，定义了一些变量和常量，其中这些常量主要是描述数据库的信息。

```
private static final String DATABASE_NAME = "database";
   private static final int DATABASE_VERSION = 1;
   private static final String DIARY_TABLE_NAME = "diary";
     private static HashMap<String, String> sDiariesProjectionMap;
     private static final int DIARIES = 1;
   private static final int DIARY_ID = 2;
     private static final UriMatcher sUriMatcher;
```

在 DiaryContentProvider 中的 static 模块中，有下边的代码。

```
sUriMatcher = new UriMatcher(UriMatcher.NO_MATCH);
sUriMatcher.addURI(Diary.AUTHORITY, "diaries", DIARIES);
sUriMatcher.addURI(Diary.AUTHORITY, "diaries/#", DIARY_ID);
```

上面代码中的 UriMatcher 是匹配 URI 的一个辅助类，对上述代码的说明如下。

- sUriMatcher.addURI(Diary.AUTHORITY, "diaries", sUriMatcher.match(uri))：如果 URI 是 content://com.shiyongcontentprovider.diarycontentprovider/diaries/，那么 sUriMatcher.match(uri) 的返回值就是 DIARIES。
- sUriMatcher.addURI(Diary.AUTHORITY, "diaries/#", DIARY_ID)：如果 URI 是 content://com.shiyongcontentprovider.diarycontentprovider/diaries/id（其中后边的 ID 是一个数字），那么 sUriMatcher.match(uri)的返回值就是 DIARY_ID。

通过 UriMatcher 类可以很方便地判断一个 URI 的类型，特别是判断这个 URI 是对单个数据的请求，还是对全部数据的请求。

第 6 步：分析抽象方法的具体实现，具体从增、删、改、查的顺序说起。

1）首先来看插入的方法 insert()，具体实现代码如下所示。

```
@Override
 public Uri insert(Uri uri, ContentValues initialValues) {
  if (sUriMatcher.match(uri) != DIARIES) {
  throw new IllegalArgumentException("Unknown URI " + uri);
  }
   ContentValues values;
  if (initialValues != null) {
  values = new ContentValues(initialValues);
  } else {
  values = new ContentValues();
  }
   if (values.containsKey(Diary.DiaryColumns.CREATED) == false) {
  values.put(Diary.DiaryColumns.CREATED, getFormateCreatedDate());
  }
   if (values.containsKey(Diary.DiaryColumns.TITLE) == false) {
  Resources r = Resources.getSystem();
  values.put(Diary.DiaryColumns.TITLE, r
  .getString(Android.R.string.untitled));
  }
   if (values.containsKey(Diary.DiaryColumns.BODY) == false) {
  values.put(Diary.DiaryColumns.BODY, "");
  }
   SQLiteDatabase db = mOpenHelper.getWritableDatabase();
  long rowId = db.insert(DIARY_TABLE_NAME, DiaryColumns.BODY, values);
  if (rowId > 0) {
  Uri diaryUri= ContentUris.withAppendedId(   Diary.DiaryColumns.CONTENT_URI, rowId);
  return diaryUri;
  }
   throw new SQLException("Failed to insert row into " + uri);
  }
```



关于对上述代码的具体说明如下。

- sUriMatcher.match(uri) != DIARIES 语句：传进来的 URI 进行了判断，如果这个 URI 不是 DIARIES 类型的，那么这个 URI 就是一个非法的 URI。
- SQLiteDatabase db = mOpenHelper.getWritableDatabase() 语句：得到一个 SQLiteDatabase 的实例。
- db.insert（DIARY_TABLE_NAME, DiaryColumns.BODY, values）语句：插入一条记录到数据库中。

insert()返回的是一个 URI，而不是一个记录的 ID，所以，需要把记录的 ID 构造成一个 Uri:Uri diaryUri= ContentUris.withAppendedId(Diary.DiaryColumns.CONTENT_URI, rowId).withAppendedId()方法在下边的小知识当中详细介绍。

ContentUris 是 content URI 的一个辅助类，主要有如下 2 个常用的方法。

- public static Uri withAppendedId（Uri contentUri, long id）：负责把 ID 和 contentUri 连接成一个新的 URI。比如在例子中使用如下代码。

```
ContentUris.withAppendedId （Diary. DiaryColumns.CONTENT_URI, rowId）
```

如果 rowId 为 100，则现在的 URI 内容如下。

```
content://com.shiyongcontentprovider.diarycontentprovider/diaries/100
```

- public static long parseId（Uri contentUri）：负责把 content URI 后边的 ID 解析出来，比如现在这个 content URI 是 content://com.shiyongcontentprovider.diarycontentprovider/diaries/100，那么这个函数的返回值就是 100。

2）下面看删除方法 delete()，具体实现代码如下所示。

```
@Override
 public int delete(Uri uri, String where, String[] whereArgs) {
   SQLiteDatabase db = mOpenHelper.getWritableDatabase();
   String rowId = uri.getPathSegments().get(1);
   return db .delete(DIARY_TABLE_NAME, DiaryColumns._ID + "=" + rowId, null);
   }
```

关于对上述代码的具体说明如下。

- rowId = uri.getPathSegments().get(1)：用于得到 rowId 的值。
- getPathSegments()方法：得到一个 String 的 List，在该例子当中 uri.getPathSegments().get(1)为 rowId，如果是 uri.getPathSegments().get(0)那值就为“diaries”。
- db.delete(DIARY_TABLE_NAME, DiaryColumns._ID + "=" + rowId, null)：是标准的 SQLite 删除操作。第一个参数数据表的名字，第二个参数相当于 SQL 语句当中的 where 部分。

3）更新数据方法 update()，具体实现代码如下所示。

```
@Override
public int update(Uri uri, ContentValues values, String where,
```

```
                String[] whereArgs) {
        SQLiteDatabase db = mOpenHelper.getWritableDatabase();
        String rowId = uri.getPathSegments().get(1);
        return db.update(DIARY_TABLE_NAME, values, DiaryColumns._ID + "="
                    + rowId, null);
    }
```

在上述代码中，先得到 SQLiteDatabase 的实例，然后得到 rowId，最后再调用 db.update(DIARY_ TABLE_NAME, values, DiaryColumns._ID + "="+ rowId, null)语句执行更新工作。

4）查寻一条数据方法 query()，具体实现代码如下所示。

```
    @Override
    public Cursor query(Uri uri, String[] projection, String selection,
                String[] selectionArgs, String sortOrder) {
        SQLiteQueryBuilder qb = new SQLiteQueryBuilder();

        switch (sUriMatcher.match(uri)) {
        case DIARIES:
            qb.setTables(DIARY_TABLE_NAME);
            break;

        case DIARY_ID:
            qb.setTables(DIARY_TABLE_NAME);
            qb.appendWhere(DiaryColumns._ID + "="
                    + uri.getPathSegments().get(1));
            break;

        default:
            throw new IllegalArgumentException("Unknown URI " + uri);
        }

        String orderBy;
        if (TextUtils.isEmpty(sortOrder)) {
            orderBy = Diary.DiaryColumns.DEFAULT_SORT_ORDER;
        } else {
            orderBy = sortOrder;
        }

        SQLiteDatabase db = mOpenHelper.getReadableDatabase();
        Cursor c = qb.query(db, projection, selection, selectionArgs, null,
                    null, orderBy);
        return c;
    }
```

关于对上述代码的具体说明如下。



- SQLiteQueryBuilder：是一个构造 SQL 查询语句的辅助类。
- sUriMatcher.match(uri)：根据返回值可以判断这次查询请求时，它是请求全部数据还是某个 ID 的数据。如果返回值是 DIARIES，那么只需要执行 qb.setTables（DIARY_TABLE_NAME）语句就可以了；如果返回值是 DIARY_ID，那么还需要将 where 部分的参数设置进去，代码为 qb.appendWhere(DiaryColumns._ID + "="+ uri.getPathSegments().get(1))。
- SQLiteDatabase db = mOpenHelper.getReadableDatabase()：得到 1 个可读的 SQLiteDatabase 实例。
- Cursor c = qb.query（db, projection, selection, selectionArgs, null,null, orderBy）语句：类似于一个标准的 SQL 查询，但是此查询是 SQLiteQueryBuilder 来发起的，而不是 SQLiteDatabase 直接发起的，所以在参数方面略有不同。这个函数为 query（SQLiteDatabase db, String[] projectionIn, String selection, String[] selectionArgs, String groupBy, String having, String sortOrder, String limit），各个参数的具体说明如下。

第一个参数为要查询的数据库实例。

第二个参数是一个字符串数组，里边的每一项代表了需要返回的列名。

第三个参数相当于 SQL 语句中的 where 部分。

第四个参数是一个字符串数组，里边的每一项依次替代在第三个参数中出现的问号（?）。

第五个参数相当于 SQL 语句当中的 groupby 部分。

第六个参数相当于 SQL 语句当中的 having 部分。

第七个参数描述是怎么进行排序。

第八个参数相当于 SQL 当中的 limit 部分，控制返回的数据的个数。

在 DiaryContentProvider 类里边还有 2 个重要的方法，一个是 getType（Uri uri），另一个是 getFormateCreatedDate()。后者根据时间得到一个特定格式的字符串，比较简单。而前者是必须要重写的方法。重写 getType 方法的具体代码如下所示。

```
public String getType(Uri uri) {
    switch (sUriMatcher.match(uri)) {
    case DIARIES:
    return DiaryColumns.CONTENT_TYPE;
      case DIARY_ID:
    return DiaryColumns.CONTENT_ITEM_TYPE;
      default:
    throw new IllegalArgumentException("Unknown URI " + uri);
    }
    }
```

此方法用于返回一个所给 Uri 的指定数据的 MIME 类型。它的返回值如果以 vnd.Android.cursor.item 开头，那么就代表这个 Uri 指定的是单条数据。如果是以 vnd.Android.cursor.dir 开头的话，那么说明这个 Uri 指定的是全部数据。

第 7 步：在程序主界面单击 MENU 键，在 ActivityMain 中执行的代码如下所示。

```
@Override public boolean onCreateOptionsMenu(Menu menu) {
```

```
        super.onCreateOptionsMenu(menu);
        menu.add(0, MENU_ITEM_INSERT, 0, R.string.menu_insert);
        return true;
        }
```

这样就给菜单（MENU）添加了一项，如图 9-24 所示。



图 9-24　单击 Menu 后的运行界面

单击添加日记按钮后进入如图 9-25 所示界面。

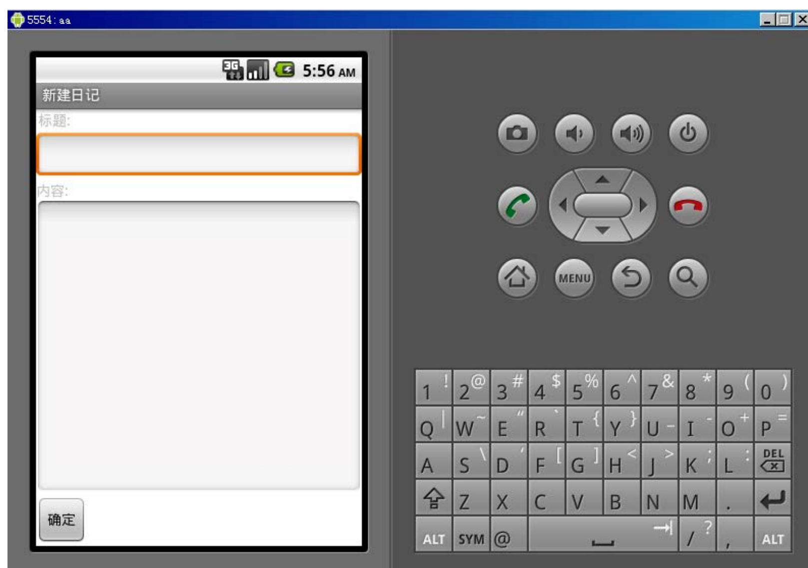


图 9-25　添加日记界面



在 ActivityDiaryEditor 中，有处理两种方法进入日记本程序界面的。一种是通过新建日记进入此界面，另一种是经过编辑日记进入此日记运行界面。下面是在 ActivityDiaryEditor 程序中使用的 onCreate()函数，其代码如下所示。

```
@Override
protected void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setTheme(android.R.style.Theme_Black);
        final Intent intent = getIntent();
        final String action = intent.getAction();
        setContentView(R.layout.diary_edit);

        mTitleText = (EditText) findViewById(R.id.title);
        mBodyText = (EditText) findViewById(R.id.body);
        confirmButton = (Button) findViewById(R.id.confirm);

        if (EDIT_DIARY_ACTION.equals(action)) {                    // 编辑日记
                mState = STATE_EDIT;
                mUri = intent.getData();
                mCursor = managedQuery(mUri, PROJECTION, null, null, null);
                mCursor.moveToFirst();
                String title = mCursor.getString(1);
                mTitleText.setTextKeepState(title);
                String body = mCursor.getString(2);
                mBodyText.setTextKeepState(body);
                setResult(RESULT_OK, (new Intent()).setAction(mUri.toString()));
                setTitle("编辑日记");
        } else if (INSERT_DIARY_ACTION.equals(action)) {        // 新建日记
                mState = STATE_INSERT;
                setTitle("新建日记");
        } else {
                Log.e(TAG, "no such action error");
                finish();
                return;
        }

        confirmButton.setOnClickListener(new View.OnClickListener() {
                public void onClick(View view) {
                        if (mState == STATE_INSERT) {
                                insertDiary();
                        } else {
                                updateDiary();
                        }
                        Intent mIntent = new Intent();
                        setResult(RESULT_OK, mIntent);
                        finish();
```

```
                }

            });

        }
```

在上述代码中，通过 getIntent()得到启动当前 Activity 的那个 Intent。

通过 intent.getAction()得到此 Intent 的动作（Action）。在此操作中，当前的 Activiy 是通过单击先前的新建日记的按钮进来的，所以，看一下 ActivityMain 当中的 onOptionsItemSelected()函数，了解一下动作（Action）是怎么设置的，代码如下所示。

```
Intent intent0 = new Intent(this, ActivityDiaryEditor.class);
intent0.setAction(ActivityDiaryEditor.INSERT_DIARY_ACTION);
intent0.setData(getIntent().getData());
startActivity(intent0);
```

从上述代码可以看到，通过语句 intent0.setAction（ActivityDiaryEditor.INSERT_ DIARY _ACTION）设置的 Action 是 ActivityDiaryEditor.INSERT_DIARY_ACTION)实现的。

在 ActivityDiaryEditor 中的 onCreate()函数中，当动作（Action）为 INSERT_DIARY_ ACTION 时候，执行 mState = STATE_INSERT，也就是标记当前的状态为插入状态。

如果当前的动作（Action）是 EDIT_DIARY_ACTION，那么当前就是编辑状态：mState = STATE_EDIT。

当运行程序填充数据后单击【确定】按钮，执行 confirmButton 的单击监听器当中的 onClick()函数。这个函数根据不同的状态执行插入或者更新数据的操作。

按照现在的程序执行流程，单击【确定】按钮后，程序将执行插入操作，实现插入操作代码如下所示。

```
private void insertDiary() {
    String title = mTitleText.getText().toString();
    String body = mBodyText.getText().toString();
    ContentValues values = new ContentValues();
    values.put(Diary.DiaryColumns.CREATED, DiaryContentProvider
        .getFormateCreatedDate());
    values.put(Diary.DiaryColumns.TITLE, title);
    values.put(Diary.DiaryColumns.BODY, body);
    getContentResolver().insert(Diary.DiaryColumns.CONTENT_URI, values);

}
```

在上述代码中，首先通过 getText()方法将编辑框中的数据都读出来。将要插入的数据都放到一个 ContentValues 的实例当中。调用 getContentResolver()得到当前应用的一个 ContentResolver 的实例。

insert（Diary.DiaryColumns.CONTENT_URI, values）语句是 ContentResolver 的插入方法，这个方法有两个参数，第一个参数是数据的 Uri；第二个参数是包含要插入数据的



ContentValues 的实例。执行这条语句的最终会调用 DiaryContentProvider 中的 insert()方法。

当单击“确定”按钮后，程序运行出现如图 9-26 所示的界面。

第 8 步：按方向键向下键将焦点移动到这条数据上，然后单击 Menu 键会出现如图 9-27 所示界面。

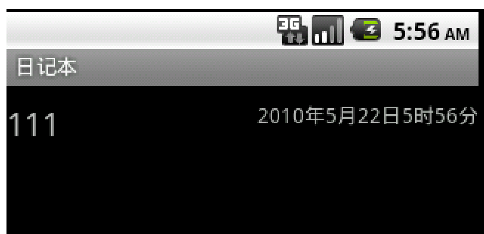


图 9-26　已经插入的数据



图 9-27　单击 Menu 键界面

图 9-27 所示的这个界面和刚才讲到的单击“Menu”键出现的界面一样，下面看一下实现的代码，以便了解此界面是怎么生成的，具体实现代码如下所示。

```
@Override
/* 在每一次 menu 生成的时候前都会调用这个方法，在这个方法里边可以动态的修改生成的 menu */
public boolean onPrepareOptionsMenu(Menu menu) {
    super.onPrepareOptionsMenu(menu);
    final boolean haveItems = getListAdapter().getCount() > 0;
    if (haveItems) {
        // 如果选中一个 Item 的话
        if (getListView().getSelectedItemId() > 0) {
            menu.removeGroup(1);
            Uri uri = ContentUris.withAppendedId(getIntent().getData(),
                    getSelectedItemId());
            Intent intent = new Intent(null, uri);
            menu.add(1, MENU_ITEM_EDIT, 1, "编辑内容").setIntent(intent);
            menu.add(1, MENU_ITEM_DELETE, 1, "删除当前日记");
        }
    }else{
        menu.removeGroup(1);
    }
    return true;
}
```

在上述代码中，和这个 ListView 相关联的 ListAdapter 里边元素的个数大于零的时候

haveItems 为真，否则为假。具体说明如下。

1）menu.removeGroup(1)：如果 menu 里边有分组为 1 组的子项的话，那么就先全部删除。

2）getIntent().getData()：得到的 URI 是 DiaryColumns.CONTENT_URI。

3）ContentUris.withAppendedId(getIntent().getData(),getSelectedItemId())：生成一个和 ListView 中获得焦点这一项的数据的 URI。

4）menu.add(1, MENU_ITEM_EDIT, 1, "编辑内容").setIntent(intent)：在 menu 当中增加了一项"编辑内容"，并且这一项和一个 Intent 关联，这个 Intent 主要用于携带数据，它里边携带的就是一个 URI 的数据，在下边的 onOptionsItemSelected()函数当中会用到。

5）menu.add(1, MENU_ITEM_DELETE, 1, "删除当前日记")：用于增加另外一项。

如果 haveItems 为假，也就是当前和这个 ListView 相关联的 ListAdapter 里边元素的个数为零，即数据显示在 ListView 上的时候，单击“Menu”按钮的话，如果 menu 当中还有第 1 分组的子项的话，就给删除了，实现语句为：menu.removeGroup(1)。

当单击“Menu”按钮时，在 Activity 中回调的函数可能有如下两个。

第一个： onOptionsItemSelected()，这个函数只在第一次在当前应用当中单击“Menu”键的时候回调，以后再不回调。

第二个：onPrepareOptionsMenu()，这个函数在每次单击“Menu”键后显示 menu 的时候被系统回调，每次 menu 显示前都会回调此函数。一般根据条件改变 menu 显示的逻辑都放在这个函数里边。

当单击“Menu”键时出现 3 个按钮，单击其中的某一个按钮会触发 Android 系统回调 onOptionsItemSelected()函数，此函数实现代码如下所示。

```
@Override public boolean onOptionsItemSelected(MenuItem item) {
  switch (item.getItemId()) {                              //插入一条数据
  case MENU_ITEM_INSERT:
  Intent intent0 = new Intent(this, ActivityDiaryEditor.class);
  intent0.setAction(ActivityDiaryEditor.INSERT_DIARY_ACTION);
  intent0.setData(getIntent().getData());
  startActivity(intent0);
  return true;                                             //编辑当前数据内容
  case MENU_ITEM_EDIT:
  Intent intent = new Intent(this, ActivityDiaryEditor.class);
       intent.setData(item.getIntent().getData());
       intent.setAction(ActivityDiaryEditor.EDIT_DIARY_ACTION);
  startActivity(intent);   return true;                    //删除当前数据
  case MENU_ITEM_DELETE:
  Uri uri = ContentUris.withAppendedId(getIntent().getData(),
  getListView().getSelectedItemId());
  getContentResolver().delete(uri, null, null);
  renderListView();
   }
  return super.onOptionsItemSelected(item);
  }
```



当用户单击添加新日记按钮后，程序新建一个跳转 Activity 的 Intent，并且设置了 Action 和 data，这两个部分在程序跳转到 ActivityDiaryEditor 后会用到。

当用户单击编辑按钮后，程序也新建了一个跳转 Activity 的 Intent，并且也设置了 Action 和 data。同样，这两部分在程序跳转到 ActivityDiaryEditor 后会用到。需要注意的是，通过 item.getIntent().getData()得到了所需要的 Uri。

当用户单击删除按钮后，程序通过 ContentUris.withAppendedId(getIntent().getData(), getListView().getSelectedItemId()) 先得到需要删除数据的 Uri，然后得到当前的 ContentResolvor，然后调用它的 delete 方法进行删除。当删除数据后，调用 renderListView() 函数对 ListView 进行及时刷新。

第 10 章　电话与短信应用开发

在本章的内容中，将进一步介绍 Intent 的使用方法，然后用电话和短信 2 个实例讲解 Android 上基本应用的开发。打电话和发短信是任何一款手机的基本功能，它需要手机平台底层（GSM/3G 模块）的支持，因此本章就通过电话和短信的例子展示应用程序该如何利用 Android 提供的 API 和设备上的通信模块打交道。

10.1　IntentFilter解析

应用程序组件为通知 Android 自己能响应、处理哪些隐式 Intent 请求，可以声明一个或多个 IntentFilter。每个 IntentFilter 描述了该组件所能响应的 Intent 请求，不能响应的将被过滤掉。IntentFilter 列出了组件希望接收什么类型的请求行为，什么类型的请求数据。例如，在请求网页浏览器时，网页浏览器程序的 IntentFilter 就应该声明它所希望接收的 Intent Action 是 WEB_SEARCH_ACTION，以及与之相关的请求数据是网页地址 URI 格式。

为组件声明自己的 IntentFilter 的常见方法是，在 AndroidManifest.xml 文件中用属性<Intent-Filter>描述组件的 IntentFilter。隐式 Intent 和 IntentFilter 进行比较时的 3 要素是 Intent 的 Action、Data 以及 Category。实际上，一个隐式 Intent 请求要能够传递给目标组件，必要通过这 3 个方面的检查。如果任何一方面不匹配，Android 都不会将该隐式 Intent 传递给目标组件。在下面的内容中，开始讲解这 3 方面检查的具体规则。

10.1.1　动作测试

<intent-filter>元素中可以包括子元素<action>，看下面的代码。

```
<intent-filter> <action android:name="com.example.project.SHOW_CURRENT" />
  <action android:name="com.example.project.SHOW_RECENT" />
  <action android:name="com.example.project.SHOW_PENDING" />
  </intent-filter>
```

一条<intent-filter>元素至少应该包含一个<action>，否则任何 Intent 请求都不能和该<intent-filter>匹配。如果 Intent 请求的 Action 和<intent-filter>中的某一条<action>匹配，那么该 Intent 就通过了这条<intent-filter>的动作测试。

如果 Intent 请求或<intent-filter>中没有说明具体的 Action 类型，那么会出现下面两种情况。

1）如果<intent-filter>中没有包含任何 Action 类型，那么无论什么 Intent 请求都无法和这条<intent-filter>匹配。

2）如果 Intent 请求中没有设定 Action 类型，那么只要<intent-filter>中包含有 Action 类型，



这个 Intent 请求就将顺利地通过<intent-filter>的行为测试。

10.1.2　类别测试

<intent-filter>元素可以包含<category>子元素，看下面的代码。

```
<intent-filter . . . >
  <category android:name="android.Intent.Category.DEFAULT" />
  <category android:name="android.Intent.Category.BROWSABLE" />
</intent-filter>
```

只有当 Intent 请求中所有的 Category 与组件中某一个 IntentFilter 的<category>完全匹配时，才会让该 Intent 请求通过测试，IntentFilter 中多余的<category>声明并不会导致匹配失败。一个没有指定任何类别测试的 IntentFilter 仅仅只会匹配没有设置类别的 Intent 请求。

10.1.3　数据测试

<intent-filter>中的数据描述如下。

```
<intent-filter . . . >
  <data android:type="video/mpeg" android:scheme="http" . . . />
  <data android:type="audio/mpeg" android:scheme="http" . . . />
  </intent-filter>
<data>
```

元素指定了希望接受的 Intent 请求的数据 URI 和数据类型，URI 被分成 3 部分来进行匹配：scheme、authority 和 path。其中，用 setData()设定的 Intent 请求的 URI 数据类型和 scheme 必须与 IntentFilter 中所指定的一致。若 IntentFilter 中还指定了 authority 或 path，它们也需要相匹配才会通过测试。

讲解完 Intent 基本概念之后，接下来就使用 Intent 激活 Android 自带的电话拨号程序。通过这个实例大家会发现，使用 Intent 并不像其概念描述得那样难。

10.2　Intent电话拨号处理

通过前面的内容，读者应该清楚了什么是 Intent，在本节的内容中，将通过一个实例来讲解如何在应用程序中使用 Intent。这里使用一个 Intent 打开电话拨号程序，Intent 的行为是 ACTION_DIAL，同时在 Intent 中传递被呼叫人的电话号码。程序的实现分为 3 个阶段，在第一阶段，只完成向固定电话拨号的工作，用户不能自由输入希望通话的电话号码。在第二阶段，再进一步完善用户界面，让用户可以自由输入电话号码，然后再拨号。在第三阶段，加入 IntentFilter，使得用户可以通过硬键盘拨号键启动程序。

本节实例保存在“光盘:\daima\10\Dialer”中，下面开始讲解其具体实现流程。

1. 基本的拨号程序

第 1 步：打开 Eclipse，新创一个名名为“Dialer”的项目。

第 2 步：设置用户界面风格。新创建的项目中用户界面默认为 Hello Android 风格（只显

示问候语字符串），因此需要修改默认的用户界面，在用户界面中加入一个 Button 按钮。编辑 res/layout/main.xml 文件，删除对<TextView>标签，加入新的<Button>标签，具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
 <LinearLayout xmlns:android=http://schemas.android.com/apk/res/android
 android:orientation="vertical"
 android:layout_width="fill_parent" android:layout_height="fill_parent"
 >
 <Button android:id = "@+id/button_id"
 android:layout_width="fill_parent"
 android:layout_height="wrap_content"
 android:text="@string/button"
 />
 </LinearLayout>
```

把 Button 的 ID 设置为 button_id，同时将 Button 显示在界面上的文字设置为 res/string.xml/下的 Button，打开 res/string.xml，把 Button 的内容设置为“拨号”。

```
<?xml version="1.0" encoding="utf-8"?>
 <resources>
 <string name="button">拨号</string>
 <string name="app_name">TinyDialer</string>
</resources>
```

第 3 步：创建 TinyDialer 的 Activity，编写 TinyDialer.java 代码，具体如下。

```
package com.studio.android.chp05.ex01;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import android.telephony.PhoneNumberUtils;

public class Dialer extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

第 4 步：定位“拨号”按钮。要加入对“拨号”按钮的响应，首先通过 findViewById()方法获得该按钮对象的引用，代码如下。



```
final Button button = (Button) findViewById(R.id.button_id);
```

第 5 步：加入对“拨号”按钮按键动作的响应。为“拨号”按钮对象调用 setOnClickListener()方法，设置单击事件监听器，代码如下。

```
final Button button = (Button) findViewById(R.id.button_id);
  button.setOnClickListener(new Button.OnClickListener() {
  @Override   public void onClick(View b) {
  //TODO 加入对按钮按下后的操作
  }
  });
```

第 6 步：建 Intent 对象，用 Intent 启动新的 Activity。此项目希望在按钮被按下后发出一个启动系统自带拨号程序的 Intent，所以首先创建 Intent 对象。创建一个新的 Intent 对象的基本语法如下。

```
Intent <Intent_name> = new Intent(<ACTION>,<Data>)
```

在本例中，参数<ACTION>为 Intent.ACTION_DIAL 参数<Data>是用户希望传入的电话号码。

在 Android 中，传给 Intent 的数据用 URI 格式表示，因此需要使用 Uri.parse 方法将字符串格式的电话号码解析成 URI 格式。在本例中，用 tel:13800138000 表示想要呼叫的电话号码。那么最终创建 Intent 的代码如下。

```
Intent I = new Intent(Intent.ACTION_DIAL,
  Uri.parse("tel://13800138000"));
```

创建 Intent 完毕后，就可以通过它告诉 Android 希望启动新的 Activity 了。

```
startActivity(i);
```

至此，一个完整的使用 Intent 的例子雏形就完成了，具体代码如下。

```
package com.studio.android.chp05.ex01;
import android.app.Activity;
 import android.content.Intent;
 import android.net.Uri;
 import android.os.Bundle;
 import android.view.View;
 import android.widget.Button;
 public class TinyDialer extends Activity {
/** Called when the Activity is first created. */
 @Override
 public void onCreate(Bundle savedInstanceState) {
 super.onCreate(savedInstanceState);
 setContentView(R.layout.main);
 final Button button = (Button) findViewById(R.id.button_id);
```

```
        button.setOnClickListener(new Button.OnClickListener() {
        @Override
        public void onClick(View b) {
        Intent i = new Intent(Intent.ACTION_DIAL, Uri.parse("tel://13800138000"));
        startActivity(i);
        }
        });
        }
       }
```

单击运行项目，此时可以看到主界面如图 10-1 所示，这个界面的布局信息都在 main.xml 文件中，在一个 LinearLayout 当中竖直排列了 5 个 Button。

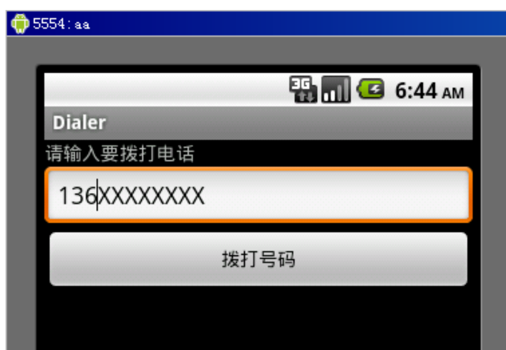


图 10-1　主界面

2. 可输入电话号码的拨号程序

下面进一步完善第一阶段的成果，使得用户能输入电话号码。由于用户输入的内容可能不是一个有效的电话号码，所以程序还需要对用户输入的字符串进行判断，呼叫有效号码，如果是无效号码则提示用户重新输入。

第 1 步：修改用户界面，加入获取用户输入的 EditText 控件。

在 Button 控件前加入一个 EditText 控件用于获取用户输入电话号码，设置其 ID 引用名为 phonenumber_id。

```
<EditText android:id = "@+id/phonenumber_id"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content"
  />
  <Button android:id = "@+id/button_id"
  android:layout_width="fill_parent"
  android:layout_height="wrap_content" android:text="@string/button"
  />
```

第 2 步：获得 EditText 对象的引用，代码如下。

```
final EditText phoneNumber =   (EditText)findViewById(R.id.phonenumber_id);
```



第 3 步：在回调方法 onClick 中加入对电话号码有效性的判断和处理，具体代码如下。

```
@Override    public void onClick(View b) {
    String callee = phoneNumber.getText().toString();
    if (PhoneNumberUtils.isGlobalPhoneNumber(callee)){
    Intent i = new Intent(Intent.ACTION_DIAL,    Uri.parse(tel"://"+ callee));    startActivity(i);
    } else {
    Toast.makeText(TinyDialer.this, R.string.notify_incorrect_phonenumber,
    Toast.LENGTH_LONG).show();
    }
    }
```

在上述代码中有以下几点需要说明。

1）判断电话号码的有效性可以使用 android.telephony.PhoneNumberUtils 包中的 isGlobalPhoneNumber 方法，Android 已经为用户准备了很多诸如此类的基本方法简化程序员的工作量，用好这些方法能够帮助用户轻松快速地完成工作。

2）对于无效电话号码的提示，可以使用 Toast 类。SDK 文档中明确说明 Toast 是包含给用户快速提醒信息的一种通知机制。当然，大家可能会问，怎么知道 Android SDK 提供了哪些机制帮助实现不同的需求？一个好办法是查看模拟器中自带的 API Demos（API 演示），如图 10−2 所示。

API Demos 已经分门别类地为开发人员组织了很多实例，这些丰富多彩的实例是开发人员获取灵感的好地方。比如这里用到的 Toast 就在 App->Notification->NotifyWithText 这个例子中。

至此，整个实例编写完毕。整理完整的 Dialer.java 代码如下。

```
package com.studio.android.chp05.ex01;

import android.app.Activity;
import android.content.Intent;
import android.net.Uri;
import android.os.Bundle;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;
import android.telephony.PhoneNumberUtils;

public class Dialer extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
```

```
        final EditText phoneNumber = (EditText) findViewById(R.id.phonenumber_id);
        final Button button = (Button) findViewById(R.id.button_id);
        button.setOnClickListener(new Button.OnClickListener()
            {
                @Override
                public void onClick(View b) {
                    String callee = phoneNumber.getText().toString();
                    if (PhoneNumberUtils.isGlobalPhoneNumber(callee)){
                        //Intent i = new Intent(Intent.ACTION_DIAL, Uri.parse("tel://" + callee));
                        Intent i = new Intent(Intent.ACTION_CALL, Uri.parse("tel://" + callee));
                        startActivity(i);
                    } else {
                        Toast.makeText(Dialer.this, R.string.notify_incorrect_phonenumber,
                            Toast.LENGTH_LONG).show();
                    }
                }
            }
        );
    }
}
```

执行后可以在框中输入要拨打的号码，输入完毕后按下“拨号”键后将看到用户界面切换到 Android 自带的拨号程序，同时所拨打的电话号码将会显示在屏幕上，如图 10-2 所示。

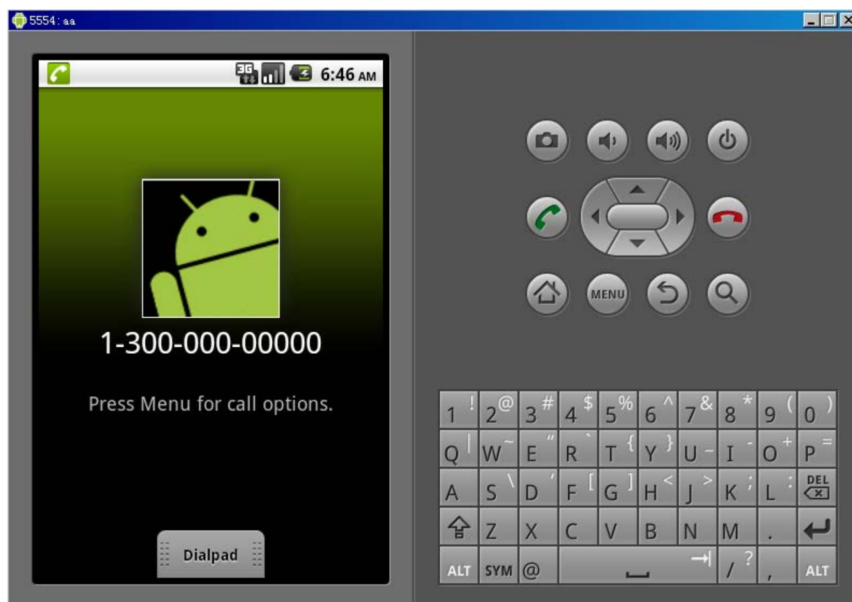


图 10-2　Android 自带拨号程序界面



3. 使用IntentFilter的拨号程序

根据之前对于 IntentFilter 的描述，从硬件键盘的拨号键启动程序需要在 TinyDialer 中加入一条新的 IntentFilter。先来看看在 AndroidManifest.xml 中关于 IntentFilter 的描述是什么。

```
<intent-filter>
<action android:name="android.Intent.Action.MAIN" />
  <category android:name="android.Intent.Category.LAUNCHER" />
</intent-filter>
```

目前只有一条 IntentFilter，它的动作名称是 Action.MAIN，类别名称是 Category. LAUNCHER。正是有了这条 IntentFilter，TinyDialer 的图标才出现在了应用程序选择的菜单里。为了新加入拨号键启动 TinyDialer，加入下面的代码。

```
<intent-filter>
  <action android:name="android.Intent.Action.CALL_BUTTON"/ >
<category android:name="android.Intent.Category.DEFAULT" />
</intent-filter>
```

更改后的效果怎么样？当按下键盘左下角绿色的拨号键时，系统会弹出一个窗口提醒用户，选择启动 TinyDialer 还是选择 Android 自带的拨号程序如图 10-3 所示。

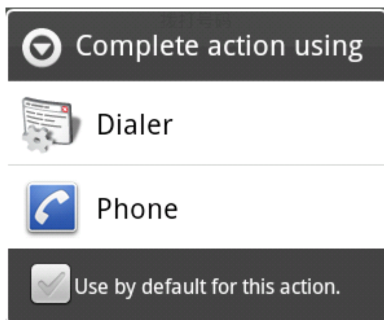


图 10-3　Android 自带拨号程序界面

这个例子很好地说明了隐式 Intent 的用法。TinyDialer 声明自己的 IntentFilter 的行为是 ACTION.CALL_BUTTON，以后每次用户按下拨号键时，Android 系统都会将拨号键的意图和所有声明过 ACTION.CALL_BUTTON 的 IntentFilter 进行比较，然后将匹配的组件提供给用户选择。

10.3　Intent短信处理

和电话拨号程序一样，短信也是任何一款手机不可或缺的基本应用，是使用频率最高的程序之一。现在，再实现一个自己的短信程序 TinySMS。这个例子中不是简单地使用 Intent 激活 Android 自带的短信程序，而是使用 SmsManager 类完成发送短信的功能。希望借此例帮助读者进一步理解 Android 中常见类的用法。由于已经有前面的拨号程序的例子，在短信程序中，将用更简洁的方式来描述开发的整个过程。

本节实例保存在“光盘:\daima\10\SMSchuli”中，下面开始讲解其具体实现流程。

10.3.1 创建TinySMS界面

第 1 步：打开 Eclipse，新创一个名名为“SMSchuli”的项目。

第 2 步：修改用户界面和 AndroidManifest.xml 文件。用户界面的 res/layout/main.xml 文件的具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
     <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="输入短信号码"
        />
    <EditText
        android:id="@+id/txtPhoneNo"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        />
    <TextView
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="短信"
        />
    <EditText
        android:id="@+id/txtMessage"
        android:layout_width="fill_parent"
        android:layout_height="150px"
        android:gravity="top"
        />
    <Button
        android:id="@+id/btnSendSMS"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="发送短信"
        />
</LinearLayout>
```

设计后的界面如图 10-4 所示。





图 10-4　程序界面

10.3.2　设置权限

因为项目程序需要使用发送短信功能，根据对 AndroidManifest.xml 的描述，需要在该文件中声明程序的权限。因此这里需要加入 TinySMS 发送短信的权限的声明。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.studio.android.chp05.ex02"
      android:versionCode="1"
      android:versionName="1.0.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".SMSchuli"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
    </application>
<uses-permission android:name="android.permission.SEND_SMS" />
</manifest>
```

在上述代码中，“<uses-permission android:name="android.permission.SEND_SMS" />”是 TinySMS 发送短信的权限的声明。

10.3.3　发送短信处理

在“发送短信”按钮的单击事件处理的回调方法 onClick()的实现中，实现发送短信的功能。具体代码如下所示。

```
            btnSendSMS.setOnClickListener(new View.OnClickListener()
            {
                public void onClick(View v)
                {
                    String phoneNo = txtPhoneNo.getText().toString();
                    String message = txtMessage.getText().toString();
                    if (phoneNo.length()>0 && message.length()>0){
                      Log.v("ROGER", "will begin sendSMS");
                            sendSMS(phoneNo, message);
                    }
                    else
                        Toast.makeText(SMSchuli.this,
                            "请重新输入电话号码和短信内容",
                            Toast.LENGTH_LONG).show();
                }
            });
        }
```

TinySMS 并不是使用 Intent 激活 Android 自带的短信程序，而是直接使用了一个叫做 sendSMS 的方法，该方法的实现代码如下所示。

```
        private void sendSMS(String phoneNumber, String message)
        {
            PendingIntent pi = PendingIntent.getActivity(this, 0,
                new Intent(this, SMSchuli.class), 0);
            Log.v("ROGER", "will init SMS Manager");
            SmsManager sms = SmsManager.getDefault();

            Log.v("ROGER", "will send SMS");
            sms.sendTextMessage(phoneNumber, null, message, pi, null);
        }
```

SmsManager 是 android.telephony.gsm.SmsManager 中定义的用户管理短信应用的类。它的用法有点特殊，开发人员不用直接实例化 SmsManager 类，而只需要调用静态方法 getDefault()获得 SmsManger 对象，方法 sendTextMessage()用于发送短信到指定号码。在上面这段代码中，使用了一个 PendingIntent 的对象，该对象指向 TinySMSActivity。因此当用户按下“发送短信”键之后，用户界面会重新回到 TinySMS 的初始界面。

在 Android 的模拟器中对短信或电话提供了非常方便的测试功能。用户只需要在 Windows 命令行中输入 emulator 再启动一个 Android 模拟器，就可以实现两个手机间的通话或者短信的测试。需要说明的是，每个模拟器左上角的数字代表了该模拟器的电话号码。

10.4 Android中电话和短信服务的包

除了在实例中包含的内容以外，复杂的电话或短信应用可以参考 Android 的相关包，它



们分别是 android.telephony 和 android.telephony.gsm。android.telephony 包中有如下类。

- CellLocation：设备位置的抽象类。
- PhoneNumberFormattingTextWather：用于监视一个 TextView 控件，
- PhoneNumberUtils：包含处理各种号码字符串的实用工具。
- PhoneStarteListener：监视手机电话状态变化的监听类。
- ServiceState：包括了电话状态和相关的服务信息。
- SMSManager：表示具体短信。
- TelephoneManager：提供对手机中电话服务信息的访问。

跟短信服务相关的类主要在包 android.telephony.gsm 中，包含了下面的类。

- GSMCellLocation：GSM 手机的基站位置。
- GSMManager：管理各种短信操作。
- GSMManage：表示具体的短信。

SmsManager 是 android.telephony.gsm.SmsManager 中定义的用户管理短信应用的类。它的用法有点特殊，开发人员不用直接实例化 SmsManager 类，而只需要调用静态方法 getDefault()获得 SmsManger 对象，方法 sendTextMessage()用于发送短信到指定号码。在上面这段代码中，使用了一个 PendingIntent 的对象，该对象指向一个 Activity 对象。因此当用户按下“发送短信”键之后，用户界面会重新回到这个 Activity 的初始界面。

第 11 章　Service的全新服务

很多情况下，一些与用户很少产生交互的应用程序，一般让它们在后台运行就行了，而且在它们运行期间仍然能运行其他的应用。为了处理这种后台进程，Android 引入了 Service 的概念。Service 在 Android 中是一种长生命周期的组件，它不实现任何用户界面。例如，最常见的媒体播放器程序，它可以在转到后台运行的时候仍然保持播放歌曲。在本章的内容中，将进一步详细讲解 Service 的基本使用方法。

11.1　Service深入剖析

什么时候需要 Service 呢？比如播放多媒体的时候用户启动了其他 Activity，这个时候程序要在后台继续播放，比如检测 SD 卡上文件的变化，再或者在后台记录用户地理信息位置的改变等，总之服务总是藏在后头的。

11.1.1　Service策略

Android 中的服务，和通常说的 Windows 服务，Web 的后台服务又有一些相近，它们通常都是后台长时间运行，接受上层指令，完成相关事务的模块。用运行模式来看，Activity 是跳，从一个跳到另一个，这样极像模态对话框（或者还像 Web 页面），给一个输入（抑或没有），然后不管不顾的让它运行，离开时返回输出（同抑或没有）。

而 Service 不是，它等着上层连接上它，然后产生一段持久而缠绵的通信，这就像一个用了 Ajax 页面，看着没啥变化，其和 Service 不知交互多少回了。

但和一般的 Service 还是有所不同，Android 的 Service 和所有 4 大组件一样，其进程模型都是可以配置的，调用方和发布方都可以有权利来选择是把这个组件运行在同一个进程下，还是不同的进程下。它凸显了 Android 的运行特征。如果一个 Service，是期望运行在调用方不同进程的时候，就需要利用 Android 提供的 RPC 机制，为其部署一套进程间通信的策略。

11.1.2　创建 Service

Android 中已经定义了一个“Service”类，所有其他的 Service 都继承于该类。Service 类中定义了一系列的生命周期相关的方法，例如，onCreate()，onStart()，onDestroy()。看下面的代码。

```
package com.wissen.testApp.service;
public class MyService extends Service {
    @Override
    public IBinder onBind(Intent intent) {
```



```
            return null;
        }
        @Override
        public void onCreate() {
            super.onCreate();
            Toast.makeText(this, "Service created…", Toast.LENGTH_LONG).show();
        }
        @Override
        public void onDestroy() {
            super.onDestroy();
            Toast.makeText(this, "Service destroyed…", Toast.LENGTH_LONG).show();
        }
    }
```

在上面的代码中，这个 Service 的功能是当服务创建和销毁时通过界面消息提示用户。

如 Android 中的其他部件一样，Service 也会和一系列 Intent 关联。Service 的运行入口需要在 AndroidManifest.xml 中进行配置，代码如下所示。

```
<service class=".service.MyService">
intent-filter>
<action android:value="com.wissen.testApp.service.MY_SERVICE" />
</intent-filter>
</service>
```

这样 Service 就可以被其他代码所使用了。

11.1.3 使用Service

应用程序可以通过调用 Context.startService 方法来启动 Service。如果当前没有指定的 Service 实例被创建，则该方法会调用 Service 的 onCreate 方法来创建一个实例；否则调用 Service 的 onStart 方法。看下面的代码。

```
Intent serviceIntent = new Intent();
serviceIntent.setAction("com.wissen.testApp.service.MY_SERVICE");
startService(serviceIntent);
```

以上代码调用了 startService 方法，Service 会持续运行，直到调用 stopService()或 stopSelf()方法。还有另一种绑定 Service 的方式。

```
ServiceConnection conn = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        Log.i("INFO", "Service bound ");
    }

    @Override
    public void onServiceDisconnected(ComponentName arg0) {
        Log.i("INFO", "Service Unbound ");
```

```
        }
    };

    bindService(new Intent("com.wissen.testApp.service.MY_SERVICE"), conn, Context.BIND_AUTO_CREATE);
```

当应用程序绑定一个 Service 后，该应用程序和 Service 之间就能进行互相通信，通常，这种通信的完成依靠于定义的一些接口，再看下面的代码。

```
package com.wissen.testApp;
public interface IMyService {
    public int getStatusCode();
}
```

这里定义了一个方法来获取 Service 的状态，但是 Service 是如何来使它起作用的呢？之前大家看到 Service 中有个返回 IBinder 对象的 onBind 方法，这个方法会在 Service 被绑定到其他程序上时被调用，而这个 IBinder 对象和之前看到的 onServiceConnected 方法中传入的那个 IBinder 是同一个东西。在实际应用中，通过 IBinder 对象和 Service 进行通信。

```
public class MyService extends Service {
    private int statusCode;
    private MyServiceBinder myServiceBinder = new MyServiceBinder();
    @Override
    public IBinder onBind(Intent intent) {
        return myServiceBinder;
    }
    public class MyServiceBinder extends Binder implements IMyService {
        public int getStatusCode() {
            return statusCode;
        }
    }
}
```

下列代码是说明了 getStatusCode 是如何被调用的。

```
ServiceConnection conn = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        IMyService myService = (IMyService) service;
        statusCode = myService.getStatusCode();
        Log.i("INFO", "Service bound ");
    }
};
```

另外，也可以通过使用 ServiceListener 接口来达成相同的目的。

11.1.4 与远程Service通信

如果两个进程间的 Service 需要进行通信，则需要把对象序列化后进行互相发送。Android



提供了一个 AIDL(Android 接口定义语言)工具来处理序列化和通信。这种情况下 Service 需要以 aidl 文件的方式提供服务接口，aidl 工具将生成一个相应的 Java 接口，并且在生成的服务接口中包含一个功能调用的 stub 服务桩类。Service 的实现类需要去继承这个 stub 服务桩类。Service 的 onBind 方法会返回实现类的对象，之后就可以使用它了，看下面的例子。

1）创建一个 IMyRemoteService.aidl 文件，内容如下。

```
package com.wissen.testApp;
interface IMyRemoteService {
    int getStatusCode();
}
```

2）如果正在使用 Eclipse 的 Android 插件，则它会根据这个 aidl 文件生成一个 Java 接口类。生成的接口类中会有一个内部类 Stub 类，用户要做的事就是去继承该 Stub 类。

```
package com.wissen.testApp;

class RemoteService implements Service {
    int statusCode;

    @Override
    public IBinder onBind(Intent arg0) {
        return myRemoteServiceStub;
    }
    private IMyRemoteService.Stub myRemoteServiceStub = new IMyRemoteService.Stub() {
        public int getStatusCode() throws RemoteException {
            return 0;
        }
    };
}
```

3）当客户端应用连接到这个 Service 时，onServiceConnected 方法将被调用，客户端就可以获得 IBinder 对象。参看下面的客户端 onServiceConnected 方法。

```
ServiceConnection conn = new ServiceConnection() {
    @Override
    public void onServiceConnected(ComponentName name, IBinder service) {
        IMyRemoteService myRemoteService = IMyRemoteService.Stub.asInterface(service);
        try {
            statusCode = myRemoteService.getStatusCode();
       } catch(RemoteException e) {
           // handle exception
       }
        Log.i(”INFO”, “Service bound “);
    }
};
```



11.1.5 设置权限

可以在 AndroidManifest.xml 文件中使用<service>标签来指定 Service 访问的权限。

```
<service    class=".service.MyService"    android:permission="com.wissen.permission.MY_SERVICE
_PERMISSION">
        <intent-filter>
            <action android:value="com.wissen.testApp.service.MY_SERVICE" />
        </intent-filter>
</service>
```

之后应用程序要访问该 Service 的话就需要使用<user-permission>标签来指定相应的权限。

```
<uses-permission android:name="com.wissen.permission.MY_SERVICE_PERMISSION">
</uses-permission>
```

11.1.6 使用Service实例

在本小节的内容中，将通过一个实例的实现过程，来讲解使用 Service 的基本流程。本实例保存在“光盘\daima\TestServiceHolder”，下面开始介绍具体实现流程。

第 1 步：打开 Eclipse，新建一个名为“TestServiceHolder”的项目，如图 11-1 所示。

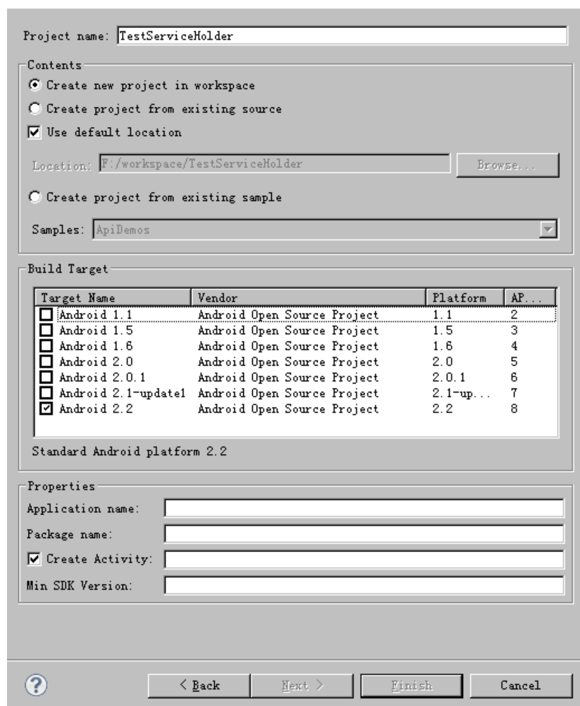


图 11-1　新建项目

第 2 步：编写主布局文件 main.xml，具体代码如下所示。



```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="@string/hello" />
    <Button android:id="@+id/start_service" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="启动  Service" />
    <Button android:id="@+id/stop_service" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="停止  Service" />
    <Button android:id="@+id/bind_service" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="Bind Service" />
    <Button android:id="@+id/unbind_service" android:layout_width="fill_parent"
        android:layout_height="wrap_content" android:text="Unbind Service" />
</LinearLayout>
```

通过上述代码，在项目中放入了 4 个按钮，分别对应要测试的 4 种操作。

第 3 步：在项目配置文件 AndroidManifest.xml 中添加对 Service 的引用，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
     package="com.iceskysl.TestServiceHolder"
     android:versionCode="1"
     android:versionName="1.0.0">
   <application android:icon="@drawable/icon" android:label="@string/app_name">
      <activity android:name=".TestServiceHolder"
               android:label="@string/app_name">
         <intent-filter>
            <action android:name="android.intent.action.MAIN" />
            <category android:name="android.intent.category.LAUNCHER" />
         </intent-filter>
      </activity>
     <service android:enabled="true" android:name=".TestService" android:process=":remote" />
   </application>
</manifest>
```

在上述代码中，通过“<service android:enabled="true" android:name=".TestService" android:process=":remote" />”实现了对 Service 的引用。

第 4 步：编写 TestService.java，具体代码如下。

```
package com.iceskysl.TestServiceHolder;

import android.app.Notification;
import android.app.NotificationManager;
import android.app.PendingIntent;
import android.app.Service;
import android.content.Intent;
```

```
import android.os.Binder;
import android.os.IBinder;
import android.util.Log;

public class TestService extends Service {

    private static final String TAG = "TestService";
    private NotificationManager _nm;

    @Override
    public IBinder onBind(Intent i) {
        Log.e(TAG, "============> TestService.onBind");
        return null;
    }
    public class LocalBinder extends Binder {
        TestService getService() {
            return TestService.this;
        }
    }
    @Override
    public boolean onUnbind(Intent i) {
        Log.e(TAG, "============> TestService.onUnbind");
        return false;
    }
    @Override
    public void onRebind(Intent i) {
        Log.e(TAG, "============> TestService.onRebind");
    }
    @Override
    public void onCreate() {
        Log.e(TAG, "============> TestService.onCreate");
        _nm = (NotificationManager) getSystemService(NOTIFICATION_SERVICE);
        showNotification();
    }
    @Override
    public void onStart(Intent intent, int startId) {
        Log.e(TAG, "============> TestService.onStart");
    }
    @Override
    public void onDestroy() {
        _nm.cancel(R.string.service_started);
        Log.e(TAG, "============> TestService.onDestroy");
    }
    private void showNotification() {
        Notification notification = new Notification(R.drawable.face_1,
                "Service started", System.currentTimeMillis());
        PendingIntent contentIntent = PendingIntent.getActivity(this, 0,
```



```
                    new Intent(this, TestServiceHolder.class), 0);
            // must set this for content view, or will throw a exception
            notification.setLatestEventInfo(this, "Test Service",
                    "Service started", contentIntent);
            _nm.notify(R.string.service_started, notification);
        }
    }
```

在上述代码中，创建了继承与 android.app.Service 的 TestService，并重写了其 onStart、onDestroy 等方法，通过输入 LOG 的方式确定被调用的方法。并且通过 Notification 来表明 Service 的存活状态。

方法 onCreate()的功能是，在 Service 的几个生命周期中增加了打印 LOG 的语句，便于查看和调试。然后通过下面的代码让调用者得到这个 Service 并操作它。

```
public class LocalBinder extends Binder {
    TestService getService() {
        return TestService.this;
    }
}
```

第 5 步：编写文件 TestServiceHolder.java，具体代码如下所示。

```
package com.iceskysl.TestServiceHolder;

import android.app.Activity;
import android.content.ComponentName;
import android.content.Context;
import android.content.Intent;
import android.content.ServiceConnection;
import android.os.Bundle;
import android.os.IBinder;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;

public class TestServiceHolder extends Activity {
    private boolean _isBound;
    private TestService _boundService;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        setTitle("Service Test");
```

```
        initButtons();

    }

    private ServiceConnection _connection = new ServiceConnection() {
        public void onServiceConnected(ComponentName className, IBinder service) {
            _boundService = ((TestService.LocalBinder)service).getService();

            Toast.makeText(TestServiceHolder.this, "Service connected",
                    Toast.LENGTH_SHORT).show();
        }

        public void onServiceDisconnected(ComponentName className) {
            //如果连接失败
            _boundService = null;
            Toast.makeText(TestServiceHolder.this, "Service connected",
                    Toast.LENGTH_SHORT).show();
        }
    };

    private void initButtons() {
        Button buttonStart = (Button) findViewById(R.id.start_service);
        buttonStart.setOnClickListener(new OnClickListener() {
            public void onClick(View arg0) {
                startService();
            }
        });

        Button buttonStop = (Button) findViewById(R.id.stop_service);
        buttonStop.setOnClickListener(new OnClickListener() {
            public void onClick(View arg0) {
                stopService();
            }
        });

        Button buttonBind = (Button) findViewById(R.id.bind_service);
        buttonBind.setOnClickListener(new OnClickListener() {
            public void onClick(View arg0) {
                bindService();
            }
        });

        Button buttonUnbind = (Button) findViewById(R.id.unbind_service);
        buttonUnbind.setOnClickListener(new OnClickListener() {
            public void onClick(View arg0) {
                unbindService();
            }
```



```
            });
        }

        private void startService() {
            Intent i = new Intent(this, TestService.class);
            this.startService(i);
        }

        private void stopService() {
            Intent i = new Intent(this, TestService.class);
            this.stopService(i);
        }

        private void bindService() {
            Intent i = new Intent(this, TestService.class);
             bindService(i, _connection, Context.BIND_AUTO_CREATE);
             _isBound = true;
        }

        private void unbindService() {
            if (_isBound) {
                unbindService(_connection);
                _isBound = false;
            }
        }
    }
```

上面的 TestServiceHolder 是个常见的 Activity，在其 onCreate 中，设定其对应的布局模板文件为“main.xml”，并调用 setTitle 设定其标题为“Service Test”。在上述代码中，使用了 Start 和 Bind 两种启动方式，当然读者也可以通过 Intent 来调用，在 Intent 中指明要启动的 Service 的名字

至此，整个实例介绍完毕，执行后的效果如图 11-2 所示。单击“启动 Service”按钮后，Service 将会启动，如图 11-3 所示。注意左上角的状态标志，表示 Service 启动了。



图 11-2 初始效果



图 11-3 Service 启动后

如果继续单击另外 3 个按钮，会发现状态标志的变化。

11.2　Service生命周期

Service 的生命周期方法比 Activity 少一些，只有 onCreate、onStart 和 onDestroy。有两种方式启动一个 Service，它们对 Service 生命周期的影响是不一样的。

1. 通过startService

Service 会经历 onCreate→onStart →stopService 的时候直接 onDestroy。

如果是调用者(TestServiceHolder)自己直接退出而没有调用 stopService 的话，Service 会一直在后台运行。下次 TestServiceHolder 再启动时可以调用 stopService。

2. 通过bindService

Service 只会运行 onCreate，此时 TestServiceHolder 和 TestService 绑定在一起。如果 TestServiceHolder 退出了，Srevice 就会调用 onUnbind→onDestroyed，这就是所谓绑定在一起就共存亡了。

另外，如果这几个方法交织在一起的话，可以遵循如下原则。

Service 的 onCreate 的方法只会被调用一次，就是无论多少次的 startService 又 bindService，Service 只被创建一次。如果先是 Bind 了，那么 Start 的时候就直接运行 Service 的 onStart 方法，如果先是 Start，那么 Bind 的时候就直接运行 onBind 方法。如果先 Bind 上了，就 Stop 不掉了，就是 stopService 不好使了，只能先 unbindService，再 stopService，所以是先 Start 还是先 Bind 行为是有区别的。

注意：onCreate 和 onStart 的不同

startService(Intent)启动。如果 service 还没有运行，则 Android 先调用 onCreate 方法然后调用 onStart 方法。如果 service 已经运行，则通过一个新的 Intent 调用 onStart 方法。所以，一个 service 的 onStart 方法可能会重复调用多次。

11.3　提高 Service 优先级

Android 系统对于内存管理有自己的一套方法，为了保障系统有序稳定的运行，系统内部会自动分配，控制程序的内存使用。当系统觉得当前的资源非常有限的时候，为了保证一些优先级高的程序能运行，就会杀掉一些它认为不重要的程序或者服务来释放内存。这样就能保证真正对用户有用的程序仍然在运行。如果 Service 碰上了这种情况，多半会先被杀掉。但如果增加 Service 的优先级就能让它多留一会，可以用 setForeground(true) 来设置 Service 的优先级。

为什么 foreground 默认启动的 Service 是被标记为 background，当前运行的 Activity 一般被标记为 foreground，也就是说给 Service 设置了 foreground，那么它就和正在运行的 Activity 类似优先级得到了一定的提高。当然，这并不能保证 Service 永远不被杀掉，只是提高了它的优先级。

有一个方法可以更清晰的演示，进入$SDK/tools 运行命令后会返回的一大堆东西，观察



oom_adj 的值，如果是大于“8”，一般就是属于 backgroud 随时可能被干掉，数值越小证明优先级越高，被干掉的时间越晚。例如 phone 的程序是-12 说明电话就是电话，其他什么都干了了，也得能接电话对吧。另外还有一个-100 的，如果 system 如果也完蛋了，系统也就崩溃了。

11.4　Service综合实例

通过前面内容的学习，了解了 Service 的基本知识和基本用法。在本节的内容中，通过一个实例的实现过程，来阐述 Service 的具体使用流程。本实例保存在“光盘:\daima\11\PlayService”，下面开始介绍其具体实现流程。

第 1 步：创建一个名为“PlayService”的 Android 工程，如图 11-4 所示。

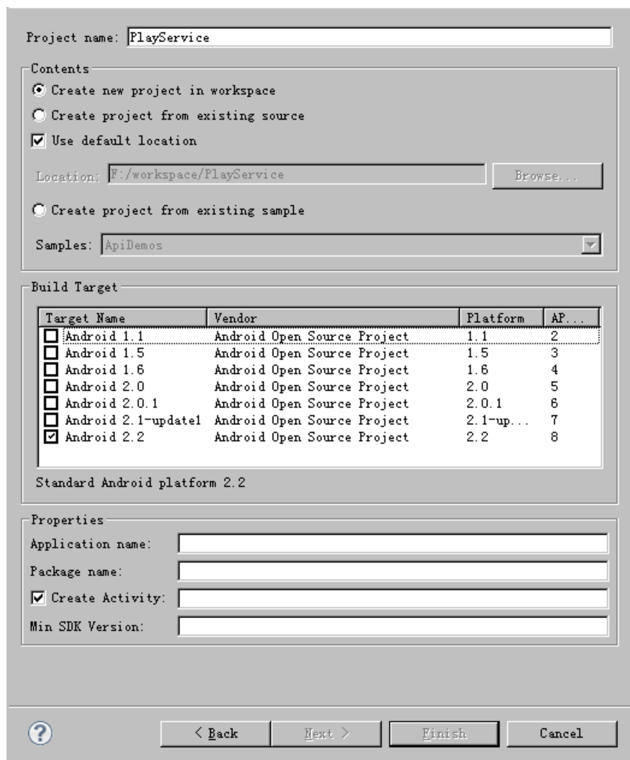


图 11-4　新建工程

第 2 步：编写主布局模板文件 main.xml，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
```




```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />

    <Button
    android:id="@+id/start"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="开始播放"
    />

    <Button
    android:id="@+id/stop"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="停止播放"
    />
</LinearLayout>
```

第 3 步：编写播放处理文件 PlayService.java，具体代码如下所示。

```
package com.iceskysl.PlayService;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;

public class PlayService extends Activity {
    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        Button button1 = (Button)findViewById(R.id.start);
        button1.setOnClickListener(startIt);
        Button button2 = (Button)findViewById(R.id.stop);
        button2.setOnClickListener(stopIt);
    }

    private OnClickListener startIt = new OnClickListener()
    {
        public void onClick(View v)
```



```
            {
                startService(new Intent("com.iceskysl.PlayService.START_AUDIO_SERVICE"));
            }
        };

        private OnClickListener stopIt = new OnClickListener()
        {
            public void onClick(View v)
            {
                stopService(new Intent("com.iceskysl.PlayService.START_AUDIO_SERVICE"));
                finish();
            }
        };
    }
```

在上述代码中，通过代码“startService(new Intent("com.iceskysl.PlayService.START_AUDIO_SERVICE"))”来启动了指定名字的服务。

第 4 步：编写处理文件 Music.java，主要代码如下所示。

```
package com.iceskysl.PlayService;

import android.app.Service;
import android.content.Intent;
import android.media.MediaPlayer;
import android.os.IBinder;

public class Music extends Service {
    private MediaPlayer player;
     @Override
     public IBinder onBind(Intent intent) {
          // TODO Auto-generated method stub
          return null;
     }

     public void onStart(Intent intent, int startId) {
        super.onStart(intent, startId);
        player = MediaPlayer.create(this, R.raw.gequ);
        player.start();
    }
    public void onDestroy() {
        super.onDestroy();
        player.stop();
    }
}
```

通过上述代码，创建了一个名为“Music”的 Service。具体过程是先创建了一个 MediaPlayer

对象 player，然后在 onStart 中播放指定的音频文件。

第 5 步：编写文件 AndroidManifest.xml，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
        package="com.iceskysl.PlayService"
        android:versionCode="1"
        android:versionName="1.0.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".PlayService"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
                <service android:name=".Music">
              <intent-filter>
                  <action android:name="com.iceskysl.PlayService.START_AUDIO_SERVICE" />
                  <category android:name="android.intent.category.default" />
              </intent-filter>
        </service>
            </intent-filter>
        </activity>

    </application>
</manifest>
```

通过上述代码，添加了对上面名为“Music”的 Service。至此，整个实例介绍完毕，执行后的演示效果如图 11-5 所示。

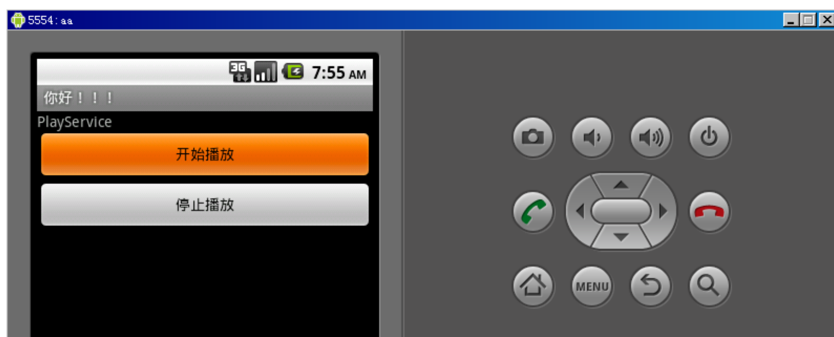


图 11-5　执行效果

单击“开始播放”按钮后，将会播放指定的音频文件。单击“停止”按钮后，会停止播放这段音频。

第 12 章　使用位置服务和地图API

Map 地图对大家来说应该不算陌生，它让人们体会到了高科技的奥妙。作为官方旗下的产品之一，Android 中可以使用 Map 地图。在官方产品中，地图功能是在 API 中的。在本章的内容中，将详细讲解 Android 中使用位置服务和地图 API 的基本流程。

12.1　位置服务

Android 支持 GPS 和网络地图，通常将各种不同的定位技术称为 LBS。LBS 是基于位置的服务（Location Based Service）的简称，它是通过电信移动运营商的无线电通信网络（如 GSM 网、CDMA 网）或外部定位方式（如 GPS）获取移动终端用户的位置信息（地理坐标，或大地坐标），在地理信息系统（Geographic Information System，GIS）平台的支持下，为用户提供相应服务的一种增值业务。

12.1.1　android.location的功能类

Android 支持地理定位服务的 API。该地理定位服务可以用来获取当前设备的地理位置。应用程序可以定时请求更新设备当前的地理定位信息。应用程序也可以借助一个 Intent 接收器来实现如下功能。以经纬度和半径划定的一个区域，当设备出入该区域时，可以发出提醒信息。在下面的内容中，开始讲解 android.location 中和定位有关的功能类。

1. Android Location API

以下是包中几个 Android 关于定位功能的比较重要的类。

- LocationManager：本类提供访问定位服务的功能，也提供获取最佳定位提供者的功能。另外，临近警报功能 （前面所说的那种功能）也可以借助该类来实现。
- LocationProvider：该类是定位提供者的抽象类。定位提供者具备周期性报告设备地理位置的功能。
- LocationListener：提供定位信息发生改变时的回调功能。必须事先在定位管理器中注册监听器对象。
- Criteria：该类使得应用能够通过在 LocationProvider 中设置的属性来选择合适的定位提供者。

2. Map API

Android 也提供了一组访问 Map 的 API，借助 Map 及定位 API，用户就能在地图上显示当前的地理位置。在 Android 中定义了一个名为 com.google.android.maps 的包，其中包含了一系列用于在 Map 上显示，控制和层叠信息的功能类，以下是该包中最重要的几个类。

- MapActivity：这个类是用于显示 Map 的 Activity 类，它需要连接底层网络。

- ❑ MapView：是用于显示地图的 View 组件，它必须和 MapActivity 配合使用。
- ❑ MapController：用于控制地图的移动。
- ❑ Overlay：这是一个可显示于地图之上的可绘制的对象。
- ❑ GeoPoint：一个包含经纬度位置的对象。

12.1.2 Android定位的基本流程

了解了 Android 中和定位处理相关的类后，下面将简要介绍在 Android 中实现定位处理的基本流程。

（1）准备 Activity 类

目标是使用 Map API 来显示地图，然后使用定位 API 来获取设备的当前定位信息以在 Map 上设置设备的当前位置，用户定位会随着用户的位置移动而发生改变。

首先需要一个继承了 MapActivity 的 Activity 类，如下面的代码。

```
class MyGPSActivity extends MapActivity {
        …
        }
```

要成功引用 Map API，还必须先在 AndroidManifest.xml 中定义如下信息。

```
<uses-library android:name="com.google.android.maps" />
```

（2）使用 MapView

要让地图显示的话，需要将 MapView 加入到应用中来。例如，在布局文件(main.xml)中加入如下代码。

```
<com.google.android.maps.MapView
        android:id="@+id/myGMap"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:enabled="true"
        android:clickable="true"
        android:apiKey="API_Key_String"
/>
```

另外，要使用 MAP 服务的话，还需要一个 API key。可以通过如下方式获取 API key。

1）找到 USER_HOME\Local Settings\Application Data\Android 目录下的 debug.keystore 文件。

2）使用 keytool 工具来生成认证信息(MD5)，使用如下命令行。

keytool -list -alias androiddebugkey -keystore <path_to_debug_keystore>.keystore -storepass android -keypass android

3）打开“Sign Up for the Android Maps API”页面，输入之前生成的认证信息(MD5)后将获取到 API key。

4）替换上面 AndroidManifest.xml 配置文件中“API_Key_String”为刚才获取的 API key。



注意：上面获取 API key 的介绍比较简单，在本章后面的内容中，将通过一个实例的实现过程来演示获取 API key 的方法。

接下来继续补全 MyGPSActivity 类的代码，在此以使用 MapView，例如下面的代码。

```
class MyGPSActivity extends MapActivity {
    @Override
    public void onCreate(Bundle savedInstanceState) {
    //创建并初始化地图
        gMapView = (MapView) findViewById(R.id.myGMap);
            GeoPoint p = new GeoPoint((int) (lat * 1000000), (int) (long * 1000000));
            gMapView.setSatellite(true);
            mc = gMapView.getController();
            mc.setCenter(p);
            mc.setZoom(14);
    }
    …
}
```

另外，如果要使用定位信息的话，必须设置一些权限，在 AndroidManifest.xml 中的具体配置如下。

```
<uses-permission android:name="android.permission.INTERNET"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_COARSE_LOCATION"></uses-permission>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"></uses-permission>
```

（3）使用定位管理器

可以通过使用 Context.getSystemService 方法，并传入 Context.LOCATION_SERVICE 参数获取定位管理器的实例。例如下面的代码。

```
LocationManager lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
```

之后，需要将原先的 MyGPSActivity 作一些修改，让它实现一个 LocationListener 接口，使其能够监听定位信息的改变。

```
class MyGPSActivity extends MapActivity implements LocationListener {
…
ublic void onLocationChanged(Location location) {}
public void onProviderDisabled(String provider) {}
public void onProviderEnabled(String provider) {}
public void onStatusChanged(String provider, int status, Bundle extras) {}
protected boolean isRouteDisplayed() {
return false;
}
}
```

下面来添加一些代码，对 LocationManager 进行一些初始化工作，并在它的 onCreate()方

法中注册定位监听器。例如下面的代码。

```
@Override
public void onCreate(Bundle savedInstanceState) {
LocationManager lm = (LocationManager)getSystemService(Context.LOCATION_SERVICE);
lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 1000L, 500.0f, this);
          }
```

此时代码中的 onLocationChanged 方法就会在用户的位置发生 500m 距离的改变之后进行调用。这里默认使用的 LocationProvider 是“gps”(GSP_PROVIDER)，但是可以根据需要，使用特定的 Criteria 对象调用 LocationManger 类的 getBestProvider 方法获取其他的 LocationProvider。以下代码是 onLocationChanged 方法的参考实现。

```
public void onLocationChanged(Location location) {
   if (location != null) {
   double lat = location.getLatitude();
   double lng = location.getLongitude();
    p = new GeoPoint((int) lat * 1000000, (int) lng * 1000000);
    mc.animateTo(p);
  }
}
```

通过上面的代码，获取了当前的新位置并更新地图上的位置显示。还可以为应用程序添加一些诸如缩放效果，地图标注，文本等功能。

（4）添加缩放控件

```
// 将缩放控件添加到地图上
ZoomControls zoomControls =   (ZoomControls) gMapView.getZoomControls();
zoomControls.setLayoutParams(new ViewGroup.LayoutParams(LayoutParams.WRAP_CONTENT,
LayoutParams.WRAP_CONTENT));
gMapView.addView(zoomControls);
gMapView.displayZoomControls(true);
```

（5）添加 Map Overlay

来到最后一步，添加 Map Overlay。通过下面的代码可以定义一个 overlay。

```
class MyLocationOverlay extends com.google.android.maps.Overlay {
@Override
 public boolean draw(Canvas canvas, MapView mapView, boolean shadow, long when) {
super.draw(canvas, mapView, shadow);
Paint paint = new Paint();
// 将经纬度转换成实际屏幕坐标
Point myScreenCoords = new Point();
mapView.getProjection().toPixels(p, myScreenCoords);
paint.setStrokeWidth(1);
paint.setARGB(255, 255, 255, 255);
paint.setStyle(Paint.Style.STROKE);
```



```
        Bitmap bmp = BitmapFactory.decodeResource(getResources(), R.drawable.marker);
        canvas.drawBitmap(bmp, myScreenCoords.x, myScreenCoords.y, paint);
        canvas.drawText("how are you…", myScreenCoords.x, myScreenCoords.y, paint);
        return true;
    }
    }
```

上面的 overlay 会在地图上显示一个“Here I am”的文本，然后把这个 overlay 添加到地图上去。

```
MyLocationOverlay myLocationOverlay = new MyLocationOverlay();
List<Overlay> list = gMapView.getOverlays();
list.add(myLocationOverlay);
```

12.1.3　GPS定位实例

本节通过一个实例的实现过程来说明使用 GPS 定位技术获取当前位置的信息。本实例代码保存在“光盘:\daima\12\”中，命名为 CurrentLocation。下面开始介绍本实例的具体实现过程。

1）在 AndroidManifest.xml 中添加 ACCESS_FINE_LOCATION 权限，具体代码如下所示。

```
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

2）编写主文件 main.xml，用于创建用户界面，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
     android:id="@+id/myLocationText"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

3）在 onCreate(Bundle savedInstanceState)中获取当前位置信息，具体代码如下所示。

```
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    LocationManager locationManager;
    String serviceName = Context.LOCATION_SERVICE;
```

```
            locationManager = (LocationManager)getSystemService(serviceName);
            //String provider = LocationManager.GPS_PROVIDER;

            Criteria criteria = new Criteria();
            criteria.setAccuracy(Criteria.ACCURACY_FINE);
            criteria.setAltitudeRequired(false);
            criteria.setBearingRequired(false);
            criteria.setCostAllowed(true);
            criteria.setPowerRequirement(Criteria.POWER_LOW);
            String provider = locationManager.getBestProvider(criteria, true);

            Location location = locationManager.getLastKnownLocation(provider);
            updateWithNewLocation(location);
            /*每隔 1000ms 更新一次，并且不考虑位置的变化。*/
            locationManager.requestLocationUpdates(provider, 2000, 10,
                    locationListener);
    }
```

在上述代码中，LocationManager 用于周期获得当前设备的一个类。要获取 LocationManager 实例，需要调用 Context.getSystemService() 方法并传入服务名 LOCATION_SERVICE("location")。创建 LocationManager 实例后，就可以通过调用 getLastKnownLocation()方法将上一次 LocationManager 获得的有效位置信息以 Location 对象的形式返回。getLastKnownLocation()方法需要传入一个字符串参数来确定使用定位服务类型，本实例传入的是静态常量 LocationManager.GPS_PROVIDER，这表示使用 GPS 技术定位。最后还需要使用 Location 对象将位置信息以文本方式显示到用户界面。

4）定义方法 updateWithNewLocation(Location location)，用于更新显示用户界面。具体代码如下所示。

```
        private void updateWithNewLocation(Location location) {
            String latLongString;
            TextView myLocationText;
            myLocationText = (TextView)findViewById(R.id.myLocationText);
            if (location != null) {
            double lat = location.getLatitude();
            double lng = location.getLongitude();
            latLongString = "纬度:" + lat + "\n 经度:" + lng;
            } else {
            latLongString = "无法获取地理信息";
            }
            myLocationText.setText("您当前的位置是:\n" +
            latLongString);
        }
    }
```

5）定义 LocationListener 对象 locationListener，当坐标改变时触发此函数。如果 Provider 传进相同的坐标，它就不会被触发。



```
private final LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {
    updateWithNewLocation(location);
    }
    public void onProviderDisabled(String provider){
    updateWithNewLocation(null);
    }
    public void onProviderEnabled(String provider){ }
    public void onStatusChanged(String provider, int status,
    Bundle extras){ }
};
```

至此整个实例介绍完毕。因为模拟器上没有 GPS 设备，所以需要在 Eclipse 的 DDMS 工具中提供模拟的 GPS 数据。即依次单击“DDMS”｜“Emulator Control”，在弹出对话框中找到“Location Control”选项，在此输入坐标，完成后单击“Send”按钮，如图 12-1 所示。



图 12-1　设置坐标

因为用到了 Google API，所以要在项目中引入 Google API，邮件单击项目选择“Properties”，在弹出对话框中选择 Google API 版本，如图 12-2 所示。

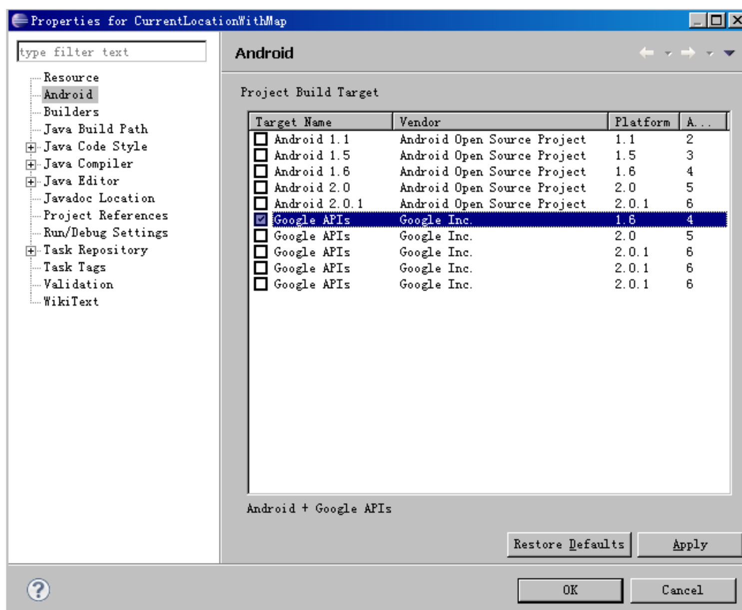


图 12-2　引用 Google API

这样模拟器运行后，会显示当前的坐标，如图 12-3 所示。



图 12-3　执行效果

12.1.4　构造LocationProvider查询条件

Android 主要通过 GPS 和网络来实现定位处理，即 GPS_PROVIDER 和 NETWORK_PROVIDER。在上一小节的实例中，显式指定了使用某种 LocationProvider 获取位置信息的方法外，还可以用程序查询 LocationProvider 条件，Android 会根据查询条件帮助程序选择最合适的提供器。在 Criteria 类中提供了如下查询条件。

- 位置解析精度，分为高或低。
- 电池消耗，分为高、中或低。
- 运营商费用。
- 海拔。
- 速度。
- 方向。

例如，下面的代码要求低位置解析精度、低电池消耗、不获取海拔高度、方向和速度的查询条件，允许运营商计算费用。

```
Criteria criteria = new Criteria();
criteria.setAccuracy(Criteria.ACCURACY_FINE);
criteria.PowerRequirment(Criteria.POWER_LOW);
criteria.setAltitudeRequired(false);
criteria.setBearingRequired(false);
criteria.setBearingRequired(Criteria.ACCURACY_FINE);
criteria.setBearingRequired(false);
criteria.setSpeedRequired(false);
criteria.setCostAllowed(true);
```

设置好上述查询条件后，可以用 getBestProvider 方法来获取和查询条件最匹配的 LocationProvider，代码如下。

```
String bestProvider = LocationProvider.getBestProvider(criteria,true);
```

如果有多个 LocationProvider 满足查询条件，那么位置解析度最高的那个提供器会最终被使用；如果没有一个匹配查询的条件，那么查询条件将变宽松，并按照下面的顺序进行新的一次查询，直到找到一个提供器为止。



- 电池消耗。
- 位置精度。
- 是否返回海拔、速度和方向。

12.2　及时监听位置变化

通过 12.1.3 中的实例，可以获取指定位置的经度、维度。但是如果 GPS 的位置信息变化后，不能及时显示新的位置信息到界面上。只有退出信息，并重新启动后才能得到最新的 GPS 信息。实际上，可以编程实现及时获位置信息，本节将简单讲解及时显示提高位置信息的方法。

12.2.1　Maps库类介绍

Maps 库提供了十几个类，具体信息可以参考如下网址。

```
http://code.google.com/intl/ja/android/add-ons/google-apis/reference/index.html
```

其中主要的类包括 MapController，MapView，MapActivity，Overlay 等。

（1）MapController

控制地图移动、伸缩，以某个 GPS 坐标为中心，控制 MapView 中的 View 组件，管理 Overlay，提供 View 的基本功能。使用多种地图模式（地图模式、卫星模式、街景模式）来查看 MAP。

常用方法有：animateTo(GeoPoint point)、setCenter(GeoPoint point)、setZoom(int zoomLevel)等。

（2）MapView

MapView 是用来显示地图的 View, 它派生自 android.view.ViewGroup。当 MapView 获得焦点，可以控制地图的移动和缩放。

地图可以以不同的形式来显示出来，如街景模式，卫星模式等，通过 setSatellite(boolean) setTraffic(boolean), setStreetView(boolean) 方法。

MapView 只能被 MapActivity 来创建，这是因为 MapView 需要通过后台的线程来连接网络或者文件系统，而这些线程要由 MapActivity 来管理。

需要特别说明的一点是，Android 1.5 中，Map 的 Zoom 采用了 built-in 机制，可以通过 setBuiltInZoomControls(boolean)来设置是否在地图上显示 Zoom 控件。

常用方法有：getController()、getOverlays()、setSatellite(boolean)、setTraffic(boolean)、setStreetView(boolean)、setBuiltInZoomControls(boolean)等。

（3）MapActivity

管理 Activity 的生命周期，为 MapView 建立及取消对 map Service 的连接。

MapActivity 是一个抽象类，任何想要显示 MapView 的 Activity 都需要派生自 MapActivity。并且在其派生类的 onCreate()中，都要创建一个 MapView 实例，可以通过 MapViewconstructor (然后添加到 View 中 ViewGroup.addView(View))或者通过 layout XML 来创建。

（4）Overlay

Overlay 是覆盖到 MapView 的最上层，可以扩展其 ondraw 接口，自定义在 MapView 中

显示一些自己的东西。MapView 通过 MapView.getOverlays()对 Overlay 进行管理。

除了 Overlay 这个基类，Google 还扩展了如下 2 个比较有用的 Overlay。

- MylocationOverlay：集成了 Android.location 中接收当前坐标的接口，集成 SersorManager 中 CompassSensor 的接口。只需要 enableMyLocation(),enableCompass 就可以让用户的程序拥有实时的 MyLocation 以及 Compass 功能（Activity.onResume()中）。
- ItemlizedOverlay：管理一个 OverlayItem 链表，用图片等资源在地图上作风格相同的标记。

（5）Projection：MapView 中 GPS 坐标与设备坐标的转换（GeoPoint 和 Point）。

12.2.2 LocationManager及时监听为遏制位置信息

LocationManager 支持监听器模式，通过调用 requestLocationUpdates()方法，能够为其设置一个位置监听器 LocationListener。同时 requestLocationUpdates()方法还需要指定要使用的位置服务类型、位置更新时间和最新位移，这样可以确保在满足用户需求的前提下最低的电量消耗。

例如，在下面的代码中，设置了更新位置信息的最小间隔为 2s，位移变化在 10m 以上。如果 GPS 位置超过 10m，且时间间隔超过 2s 时，LocationListener 的回调方法 onLocationChanged() 就会被调用，应用程序可以通过 onLocationChanged 来反映位置信息的变化。

```
public class CurrentLocationWithMap extends MapActivity {
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        LocationManager locationManager;
        String context = Context.LOCATION_SERVICE;
        locationManager = (LocationManager)getSystemService(context);
        //String provider = LocationManager.GPS_PROVIDER;
        /*Location Provider 查询条件*/
        Criteria criteria = new Criteria();
        criteria.setAccuracy(Criteria.ACCURACY_FINE);
        criteria.setAltitudeRequired(false);
        criteria.setBearingRequired(false);
        criteria.setCostAllowed(true);
        criteria.setPowerRequirement(Criteria.POWER_LOW);
        String provider = locationManager.getBestProvider(criteria, true);

        Location location = locationManager.getLastKnownLocation(provider);
        updateWithNewLocation(location);
        /*设置更新位置信息的最小间隔为 2s，位移变化在 10m 以上*/
        locationManager.requestLocationUpdates(provider, 2000, 10,
                locationListener);
    }
   /*Location  发生变化时被调用*/
   private final LocationListener locationListener = new LocationListener() {
     public void onLocationChanged(Location location) {
     updateWithNewLocation(location);
```



```
        }
        public void onProviderDisabled(String provider){
        updateWithNewLocation(null);
        }
        public void onProviderEnabled(String provider){ }
        public void onStatusChanged(String provider, int status,
        Bundle extras){ }
    };
    private void updateWithNewLocation(Location location) {
        String latLongString;
        TextView myLocationText;
        myLocationText = (TextView)findViewById(R.id.myLocationText);
        if (location != null) {
            double lat = location.getLatitude();
            double lng = location.getLongitude();
            latLongString = "纬度:" + lat + "\n 经度:" + lng;

            ctrlMap.animateTo(new GeoPoint((int)(lat*1E6),(int)(lng*1E6)));
        } else {
            latLongString = "无法获取地理信息";
        }
        myLocationText.setText("您当前的位置是:\n" +
        latLongString);

    }
```

可以将上述代码直接添加到 12.1.3 的实例中，执行后，当在 DDMS 中修改经度和维度后，会发现显示界面中的内容也随之发生变化。

12.3　应用地图

本章前面的讲解，都是用文字方式显示位置信息的。在 Android 中，是可以直接使用 Google 地图的，可以以地图的形式显示位置信息。在本节的内容中，将详细讲解在程序中应用 Google 地图的基本流程。

12.3.1　使用前的设置

Google 地图给人们的生活带来了极大的方便，例如，可以通过 Google 地图查找商户信息、查看地图和获取行车路线等。Android 平台也提供了一个 Map 包(com.google.android.maps)，通过其中的 MapView 就能够方便地利用 Google 地图的资源来进行编程。在使用前需要预先进行如下必要的设置。

（1）添加 maps.jar 到项目

在 Android SDK 中，以 JAR 库的形式提供了和 MAP 有关的 API，此 JAR 库位与“android-sdk-windows\add-ons\google_apis-4”目录下。要把 maps.jar 添加到项目中，可以在项

目属性中的“Android”栏中指定使用包含 Google API 的 Target 作为项目的构建目标，如图 12-4 所示。

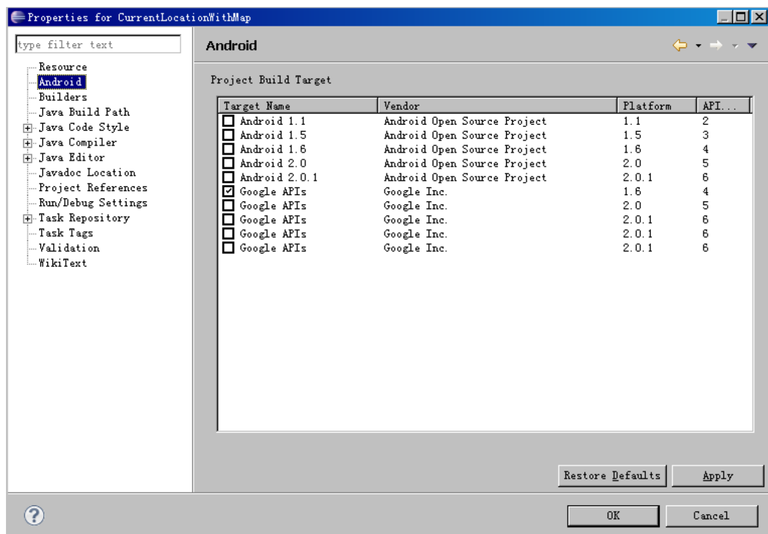


图 12-4 在项目中包含 Google API

（2）将地图嵌入到应用

通过使用 MapActivity 和 MapView 控件，可以轻松地将地图嵌入到应用程序当中。在此步骤中，需要将 Google API 添加到构建路径中。方法是在图 12-4 所示界面中选择“Java Build Path”，然后在 Target 中勾选 Google API，设置项目中包含 Google API，如图 12-5 所示。

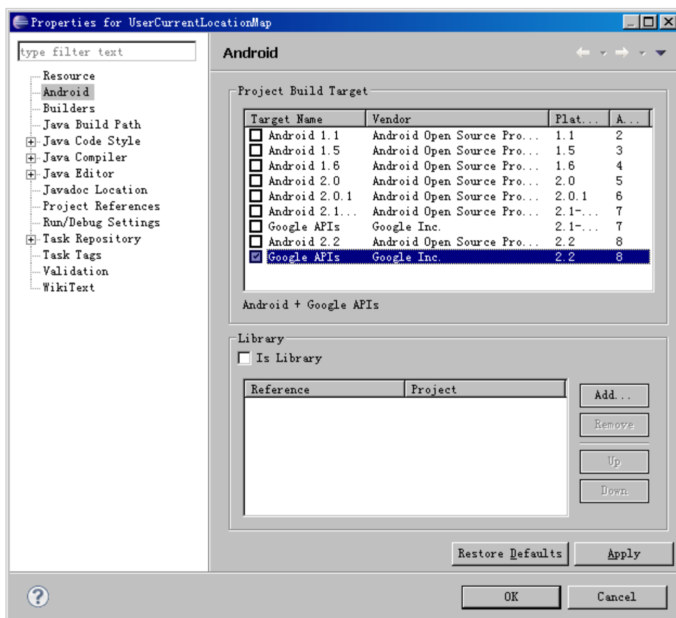


图 12-5 将 Google API 添加到构建路径



（3）获取 Map API 密钥

在利用 MapView 之前，必须要先申请一个 Android Map API Key，具体步骤如下。

第 1 步：找到你的 debug.keystore 文件，通常位于如下目录。

C:\Documents and Settings\你的当前用户\Local Settings\Application Data\Android

第 2 步：获取 MD5 指纹。运行 cmd.exe，执行如下命令获取 MD5 指纹。

```
>keytool -list -alias androiddebugkey -keystore "debug.keystore 的路径" -storepass android -keypass android
```

例如，输入如下命令。

```
keytool -list -alias androiddebugkey -keystore "C:\Documents and Settings\Administrator\.android\debug.keystore" -storepass android -keypass android
```

此时系统会提示输入 keystore 密码，这时候输入 android，系统就会输出申请到的 MD5 认证指纹，如图 12-6 所示。

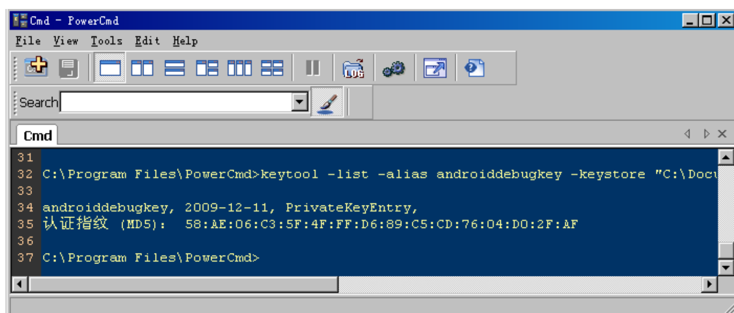


图 12-6　获取的认证指纹

第 3 步：申请 Android map 的 API Key。

打开浏览器，输入下面的网址：http://code.google.com/intl/zh-CN/ android/maps- api-signup.html，如图 12-7 所示。

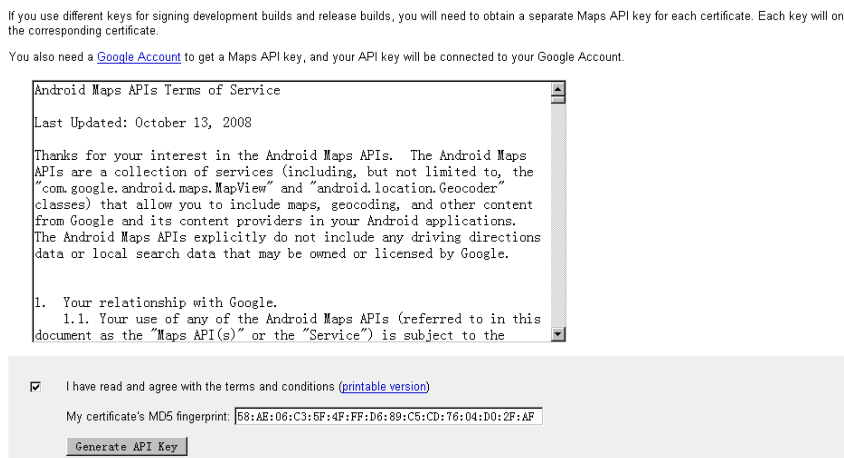


图 12-7　申请主页

在 Google 的 android map API Key 申请页面上输入图 12-6 中得到的 MD5 认证指纹，按下“Generate API Key”按钮后即可转到下面的这个画面，得到申请到的 API Key，如图 12-8 所示。



图 12-8　得到的 API Key

至此，成功地获取了一个 API Key，编程前的整个准备工作也完成了。

12.3.2　使用Map API密钥的基本流程

通过上一节的讲解，已经申请到了一个 Android Map API Key，下面开始讲解使用 Map API 密钥实现编程的基本流程。

第 1 步：在文件 AndroidManifest.xml 中声明权限。

在 Anroid 系统中，如果程序执行需要读取到安全敏感的项目，那么必须在 AndroidManifest.xml 中声明相关权限请求，比如这个地图程序需要从网络读取相关数据。所以必须声明 android.permission.INTERNET 权限。具体方法是在 AndroidManifest.xml 中添加如下代码。

```
<uses-permission android:name="android.permission.INTERNET" />
```

另外，因为 Maps 类不是 Android 启动的默认类，所以还需要在文件 AndroidManifest.xml 的 application 标签中申明要用 Maps 类。

```
<uses-library android:name="com.google.android.maps" />
```

下面是基本的 AndroidManifest.xml 文件代码。

```
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
    <application android:icon="@drawable/icon" android:label="@string/app_name">
     <uses-library android:name="com.google.android.maps" />
    </application>
  <uses-permission android:name="android.permission.INTERNET" />
</manifest>
```

第 2 步：在 main.xml 主文件中完成 Layout。



下面开始着手来完成界面。假设设置要显示杭州的卫星地图，并在地图上方有 5 个按钮，分别可以放大地图、缩小地图或者切换显示模式（卫星，交通，街景）。即整个界面主要由 2 个部分组成，上面是一排 5 个按钮，下面是 MapView。

在 Android 中，LinearLayout 是可以互相嵌套的，在此可以把上面 5 个按钮放在一个子 LinearLayout 里边(子 LinearLayout 的指定可以由 android:addStatesFromChildren="true"实现)，然后再把这个子 LinearLayout 加到外面的父 LinearLayout 里边。具体实现如下。

```
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
  android:orientation="vertical" android:layout_width="fill_parent"
  android:layout_height="fill_parent">

  <LinearLayout android:layout_width="fill_parent"
    android:addStatesFromChildren="true"            /*说明是子 Layout
    android:gravity="center_vertical"               /*这个子 Layout 里边的按钮是横向排列
    >

    <Button android:id="@+id/ZoomOut"
      android:text="放大"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:layout_marginTop="5dip"               /*下面的 4 个属性,指定了按钮的相对位置
      android:layout_marginLeft="30dip"
      android:layout_marginRight="5dip"
      android:layout_marginBottom="5dip"
      android:padding="5dip" />

    /*其余 4 个按钮省略
  </LinearLayout>
  <com.google.android.maps.MapView
    android:id="@+id/map"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:enabled="true"
    android:clickable="true"
    android:apiKey="在此输入上一节申请的 API Key"     /*必须加上上一节申请的 API Key
  />

</LinearLayout>
```

第 3 步：完成主 Java 程序代码。

首先，主文件的这个类必须继承 MapActivity。

```
public class Mapapp extends MapActivity {
```

然后，来关注 onCreate()函数，其核心代码如下。

```
  public void onCreate(Bundle icicle) {
  //取得地图 View
    myMapView = (MapView) findViewById(R.id.map);
```

```
        //设置为卫星模式
        myMapView.setSatellite(true);
        //地图初始化的点:杭州
        GeoPoint p = new GeoPoint((int) (30.27 * 1000000),
            (int) (120.16 * 1000000));
        //取得地图 View 的控制
        MapController mc = myMapView.getController();
        //定位到杭州
        mc.animateTo(p);
        //设置初始化倍数
        mc.setZoom(DEFAULT_ZOOM_LEVEL);
    }
```

接着，编写缩放按钮的处理代码，具体如下。

```
        btnZoomIn.setOnClickListener(new View.OnClickListener() {
          public void onClick(View view) {
              myMapView.getController().setZoom(myMapView.getZoomLevel() - 1);
        });
```

地图模式的切换由下面代码实现。

```
      btnSatellite.setOnClickListener(new View.OnClickListener() {
      public void onClick(View view) {
        myMapView.setSatellite(true);                    //卫星模式为 True
        myMapView.setTraffic(false);                     //交通模式为 False
        myMapView.setStreetView(false);                  //街景模式为 False
      }
    });
```

到此为止，就完成了第一个使用 Map API 的应用程序。

12.3.3 应用实例：使用Map API密钥

在本节的内容中，将通过一个实例的实现过程来讲解使用 Map API 密钥实现 Google 地图定位的基本流程。本实例源文件保存在“光盘:\daima\12\”中，命名为“CurrentLocationWithMap”。下面开始介绍本实例的具体实现流程。

（1）编写主布局文件 main.xml

在布局文件 main.xml 中，插入了 2 个 Button 按钮，分别实现对地图的“放大”和“缩小”；然后，通过 ToggleButton 控制是否显示卫星地图；最后，设置申请的 API Key。具体代码如下所示。

```
    <?xml version="1.0" encoding="utf-8"?>
    <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical"
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
```



```
        >
    <TextView
         android:id="@+id/myLocationText"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        />
    <LinearLayout
        android:orientation="horizontal"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content" >
        <Button
            android:id="@+id/in"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="放大" />
        <Button
            android:id="@+id/out"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:text="缩小" />
    </LinearLayout>
    <ToggleButton
        android:id="@+id/switchMap"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:textOff="卫星视图(关)"
        android:textOn="卫星视图(开)"/>
    <com.google.android.maps.MapView
         android:id="@+id/myMapView"
         android:layout_width="fill_parent"
         android:layout_height="fill_parent"
         android:clickable="true"
         android:apiKey="0by7ffx8jX0A_LWXeKCMTWAh8CqHAlqvzetFqjQ"
         />
    </LinearLayout>
```

（2）声明权限

在文件 AndroidManifest.xml 中，需要声明 android.permission.INTERNET 和 INTERNET 权限，具体代码如下所示。

```
<uses-permission android:name="android.permission.INTERNET"/>
<uses-permission android:name="android.permission.ACCESS_FINE_LOCATION"/>
```

（3）编写主程序文件 CurrentLocationWithMap.java

第 1 步：通过方法 onCreate 将 MapView 绘制到屏幕上。因为 MapView 只能继承自 MapActivity 的活动中，所以必须用方法 onCreate 将 MapView 绘制到屏幕上，并同时覆盖方法 isRouteDisplayed()，它表示是否需要在地图上绘制导航线路，具体代码如下所示。

```
package com.CurrentLocationWithMap;

import java.util.List;

import android.content.Context;
import android.location.Criteria;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.CompoundButton;
import android.widget.TextView;
import android.widget.ToggleButton;
import android.widget.CompoundButton.OnCheckedChangeListener;

import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import com.google.android.maps.MyLocationOverlay;
import com.google.android.maps.Overlay;

public class CurrentLocationWithMap extends MapActivity {

    MapView map;

    MapController ctrlMap;
    Button inBtn;
    Button outBtn;
    ToggleButton switchMap;

    @Override
     protected boolean isRouteDisplayed() {
          return false;
     }
```

第 2 步：定义方法 onCreate，首先引入主布局 main.xml，并通过方法 findViewById 获得 MapView 对象的引用，接着调用 getOverlays()方法获取其 Overylay 链表，并将构建好的 MyLocationOverlay 对象添加到链表中去。其中 MyLocationOverlay 对象调用的



enableMyLocation()方法表示尝试通过位置服务来获取当前的位置，具体代码如下所示。

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);

    map = (MapView)findViewById(R.id.myMapView);
    List<Overlay> overlays = map.getOverlays();
    MyLocationOverlay myLocation = new MyLocationOverlay(this,map);
    myLocation.enableMyLocation();
    overlays.add(myLocation);
```

还需要为“放大”和“缩小”这 2 个按钮设置处理程序，首先通过方法 getController()获取 MapView 的 MapController 对象，然后在“放大”和“缩小”两个按钮单击事件监听器的回放方法里，根据按钮的不同实现对 MapView 的缩放，具体代码如下所示。

```
ctrlMap = map.getController();
inBtn = (Button)findViewById(R.id.in);
outBtn = (Button)findViewById(R.id.out);
OnClickListener listener = new OnClickListener() {
    @Override
    public void onClick(View v) {
        switch (v.getId()) {
        case R.id.in:                          /*如果是缩放*/
            ctrlMap.zoomIn();
            break;
        case R.id.out:                         /*如果是放大*/
            ctrlMap.zoomOut();
            break;
        default:
            break;
        }
    }
};
inBtn.setOnClickListener(listener);
outBtn.setOnClickListener(listener);

//========================================
```

第 3 步：通过方法 onCheckedChanged 来获取是否选择了 switchMap，如果选择了则显示卫星地图。首先通过方法 findViewById 获取对应 ID 的 ToggleButton 对象的引用，然后调用 setOnCheckedChangeListener 方法，设置对事件监听器选中的事件进行处理。根据 ToggleButton 是否被选中，进而通过 setSatellite()方法启用或禁用卫星试图功能。具体代码如下所示。

```
        switchMap = (ToggleButton)findViewById(R.id.switchMap);
```

```
        switchMap.setOnCheckedChangeListener(new OnCheckedChangeListener() {
            @Override
            public void onCheckedChanged(CompoundButton cBtn, boolean isChecked) {
                if (isChecked == true) {
                    map.setSatellite(true);
                } else {
                    map.setSatellite(false);
                }
            }
        });
```

第 4 步：首先通过 LocationManager 获取当前的位置，然后通过 getBestProvider 方法来获取和查询条件，最后设置更新位置信息的最小间隔为 2s，位移变化在 10m 以上。具体代码如下所示。

```
    LocationManager locationManager;
    String context = Context.LOCATION_SERVICE;
    locationManager = (LocationManager)getSystemService(context);
    //String provider = LocationManager.GPS_PROVIDER;

    Criteria criteria = new Criteria();
    criteria.setAccuracy(Criteria.ACCURACY_FINE);
    criteria.setAltitudeRequired(false);
    criteria.setBearingRequired(false);
    criteria.setCostAllowed(true);
    criteria.setPowerRequirement(Criteria.POWER_LOW);
    String provider = locationManager.getBestProvider(criteria, true);

    Location location = locationManager.getLastKnownLocation(provider);
    updateWithNewLocation(location);
    locationManager.requestLocationUpdates(provider, 2000, 10,
            locationListener);
}
```

第 5 步：设置回调方法何时被调用，具体代码如下所示。

```
private final LocationListener locationListener = new LocationListener() {
    public void onLocationChanged(Location location) {
    updateWithNewLocation(location);
    }
    public void onProviderDisabled(String provider){
    updateWithNewLocation(null);
    }
    public void onProviderEnabled(String provider){ }
    public void onStatusChanged(String provider, int status,
    Bundle extras){ }
    };
```



第 6 步：定义方法 updateWithNewLocation(Location location)，用于显示地里信息和地图信息。具体代码如下所示。

```
private void updateWithNewLocation(Location location) {
        String latLongString;
        TextView myLocationText;
        myLocationText = (TextView)findViewById(R.id.myLocationText);
        if (location != null) {
            double lat = location.getLatitude();
            double lng = location.getLongitude();
            latLongString = "纬度:" + lat + "\n 经度:" + lng;

            ctrlMap.animateTo(new GeoPoint((int)(lat*1E6),(int)(lng*1E6)));
        } else {
            latLongString = "无法获取地理信息";
        }
        myLocationText.setText("您当前的位置是:\n" +
        latLongString);

       }
}
```

至此，整个实例介绍完毕，在图 12-9 中选定一个经度和维度位置后，可以显示此位置的定位信息，并且定位信息分别以文字和地图形式显示出来，如图 12-10 所示。

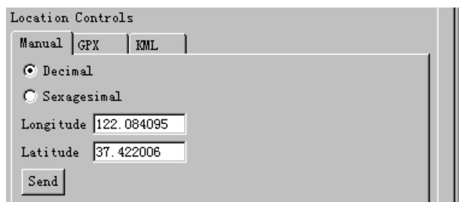


图 12-9　指定位置



图 12-10　显示对应信息

单击“放大”和“缩小”按钮后，能控制地图的大小显示，如图 12-11 所示。打开卫星

试图后，可以显示此位置范围对应的卫星地图，如图 12-12 所示。



图 12-11　放大后效果



图 12-12　卫星地图

第 13 章　程 序 优 化

作为一个程序，无论是简单还是复杂，都可能有多种编码格式，不同的编码格式肯定效率不尽相同。不知从何时起，程序优化提到了日常议程。但是事实证明，经过优化处理后的程序，其运行速度和效率大大提高了。试想作为一个访问量巨大的项目，如果每个访问速度快那么一点点，在海量访问的前提下，速度会快不少。在本章的内容中，将简要讲解优化 Android 程序和 Java 程序的基本知识。

13.1　实现高效Java编程规范的十条基础规则

编码规范对于程序员而言尤为重要，有以下几个原因。

- 一个软件的生命周期中，80%的花费在于维护。
- 几乎没有任何一个软件，在其整个生命周期中，均由最初的开发人员来维护。
- 编码规范可以改善软件的可读性，可以让程序员尽快而彻底地理解新的代码。
- 如果将源码作为产品发布，就需要确认它是否被很好地打包并且清晰无误。

为了执行规范，每个软件开发人员必须一致遵守编码规范。笔者抛砖引玉，在借用前辈理论的基础上，总结了实现高效 Java 编程规范的十条基础规则。

1）避免使用 NEW 关键字来创建 String 对象。

把一个 String 常量 copy 到 String 对象中通常是多余、浪费时间的。

```
Public class test{
  Public void method(){
    System.out.print (str);
  }
  private String str = new String ("1");          //这里新建对象是完全没有必要的
    private String str2=”2”                       //正确的应该如此
}
```

2）避免使用不必要的嵌套。

过多的嵌套会使你的代码复杂化，减弱可读性。例如：

```
Public class test {
  String add (){
    Int c=(a=a+b)+b; //过于复杂
    Return c
  }
}
```

3）避免在同一行声明不同类型的多个变量，这样可以使程序更加清晰，避免混乱。例如：

```
private int index, index1[];
```

正确的写法应该如下。

```
private int index;
private int index1[];
```

4）在每一行里写一条语句。

这条规则不包括 for 语句：比如：for (int i = 0; i < 10; i++) x−−;可以增加代码的可读性。

```
public class OSPL {
  int method (int a, int b) {
    int i = a + b; return i; //  可读性不强
  }
```

正确的写法如下。

```
public class OSPLFixed {
  int method (int a, int b) {
    int i = a + b;
    return i;
  }
}
```

5）经常从 finalize ()中调用 super.finalize ()。

这里的 finalize()是 Java 在进行垃圾收集的时候调用的，和 finally 不一样。如果父类没有定义 finally()的话，也应该调用。这里有如下两个原因。

- 在不改变代码的情况下能够将父类的 finally 方法加到类中。
- 以后会养成习惯调用父类的 finally 方法，即使父类没有定义 finally 方法的时候。

正确的方法应该如下。

```
public class parentFinalize {
  protected void finalize () throws Throwable {
    super.finalize(); // FIXED
  }
```

6）不要在 finalize ()中注销 listeners。finalize ()只有在没有对象引用的时候调用，如果 listeners 从 finalize()方法中去除了，被 finalize 的对象将不会在垃圾收集中去除。例如：

```
public void finalize () throws Throwable {
    bButton.removeActionListener (act);
}
```

7）不要显式地调用 finalize ()方法。

虽然显式地调用这个方法可以确保调用，但是当这个方法收集了以后垃圾收集会再收集



一次。例如：

```
public class T7 {
  public void finalize() throws Throwable {
    close_resources ();
    super.finalize ();
  }
  public void close_resources() {}
}
class Test {
  void cleanup () throws Throwable {
    t71.finalize(); //  调用
    t71 = null;
  }
    private t71 = new T7 ();
}
```

对于这样的调用应该自己创建一个释放的方法，做最初 finalize ()所做的事情，当用户每次想显式地调用 finalize ()的时候实际上调用了释放方法。然后再使用一个判断字段来确保这个方法只执行一次，以后再调用就没关系了。例如：

```
public class T7 {
  public synchronized void release () throws Throwable{
    if (!_released) {
      close_resources (); // do what the old 'finalize ()'
      did _released = true;
    }
  }
    public void finalize () throws Throwable {
    release ();
      super.finalize ();
  }
  public void close_resources() {}
    private boolean _released = false;
}
class TestFixed {
  void closeTest () throws Throwable {
    t71 .release (); // FIXED
    t71 = null;
  }
  private T7 t71 = new T7 ();
}
```

8）不要使用不推荐的 API。尽量使用 JDK1.3 推荐的 API。在类和方法或者 Java 组件里有很多方法是陈旧的或者是可以选择的。有一些方法 SUN 用了“deprecated”标记。建议读者最好不要使用。例如：

```
private List t_list = new List ();
t_list.addItem (str);
```

如果查 javadoc，会发现建议用 add()来代替 addItem()。

9）为所有序列化的类创建一个“serialVersionUID”。

可以避免从各种不同的类破坏序列的兼容性。如果不特别制订一个 UID，那么系统会自动产生一个 UID（根据类的内容）。如果 UID 在新版本的类中改变了，即使那个被序列化的类没改变，也不能反序列化老的版本了。例如：

```
public class DUID implements java.io.Serializable { public void method () {}}
```

在里面加一个 UID，当这个类的序列化形式改变的时候，也改变这个 UID 就可以了。例如：

```
public class DUIDFixed implements java.io.Serializable {
  public void method () {}
  private static final long serialVersionUID = 1;
}
```

10）对于 private 常量的定义，比较好的做法是对于这样的常量，加上 final 标记，这样的常量从初始化到最后结束值都不会改变。例如：

private int size = 5;

改变后的做法是

private final int size = 5;

11）避免把方法本地变量和参数定义成和类变量相同的名字。

这样很容易引起混淆，建议把任何的变量字都定义成唯一的。这样看来，SCJP 里的那些题目在现实中就用不到了。例如：

```
public void method (int j) { final int i = 5; // VIOLATION } private int j = 2;
```

建议写法如下。

```
public void method (int j1) { final int i = 5; // VIOLATION } private int j = 2;
```

13.2　命名规范

此处的命名规范包括对变量、常量、类等的命名规范。遵循科学合理的命名规范，可以提高程序的运行效率，便于程序的后期维护。本节简要讲解 Java 的命名规范。

（1）对常量的命名规范

常量名应使用大写字母，单词间用下画线隔开，并接后能见其名知其意。例如，MAX_VALUE 常量用来储存一个最大值。

（2）对变量的命名规范

变量名应小写，且要有意义，尽量避免使用单个字符，否则遇到该变量时很难理解其用



途。对于临时的变量，例如，记忆循环语句中的循环次数，通常可命名为 i，j，k 这样的单字符变量名。变量名应简短且富于描述，以便容易记忆，如用 age 变量来储存年龄。

（3）对方法的命名规范

方法被调用来执行一个操作，因此方法名应是对该操作的描述。方法名的首字母应该小写，如果由多个单词组成，则其后单词的首字母大写。例如，一个向数据库添加数据的方法可命名为 addData()。

（4）对包的命名规范

包名的前缀应该全部由小写英文字母组成，如 java.io。

（5）对类的命名规范

类名应使用单词，首字母须大写，若由多个单词组成，则每个单词的首字母大写，尽量使类名简洁而富于描述，如 RandomAccessFile。

（6）对接口的命名规范

与类的命名规范相同，如 FileFilter。当在任何需要进行命名的情况下，建议不要使用汉字或其他语言中的文字来命名，虽然在 Java 中是允许使用的。

13.3　编写优秀代码的技巧

掌握了基本的编程语法，不一定代表就是一个优秀的程序员，不代表能编写出好的代码。编写科学、合理、高效的代码，才是一个优秀程序员的标志。编写优秀代码的技巧如下。

1. 使用好的编码风格和合理的设计

可以通过采用良好的编程风格，来防范大多数编码错误。这与本篇的其他章节自然地吻合。很多简单的事，如选用有意义的变量名，或者审慎地使用括号，都可以使编码变得更加清晰明了，并降低缺陷出现的可能性。

在投入到编码工作中之前，先考虑大体的设计方案，这也非常关键。“最好的计算机程序的文本是结构清晰的。”（见参考书目 Kernighan Plaugher 78）从实现一套清晰的 API、一个逻辑系统结构以及一些定义良好的组件角色与责任开始入手，将使大家避免以后处处头疼的局面。

2. 误信第三者

即便是没有恶意的代码用户，也可能会给程序带来麻烦。防御意味着不能相信任何人。下面这些情况可能是带来麻烦的原因。

- 真正的用户：意外地提供了假的输入，或者错误地操作了程序。
- 恶意的用户：故意造成不好的程序行为。
- 客户端代码：使用错误的参数调用了你的函数，或者提供了不一致的输入。
- 运行环境：没有为程序提供足够的服务。
- 外部程序库：运行失误，不遵从所依赖的接口协议。

甚至可能会在编写一个函数时犯下愚蠢的错误，或者错误地使用 3 年前编写的代码，因为忘记了这些代码究竟是怎样运行的。不要设想所有的一切都运行良好，或者所有的代码都会正确地运行。在程序各处都添加安全检查，时刻注意弱点，用更多的防御性代码防止弱点的出现。

3. 不要仓促地编写代码

闪电式的编程太常见了。使用这种编程方式的程序员会很快地开发出一个函数，马上把这个函数交给编译器来检查语法，接着运行一遍看看能不能用，然后就进入下一个任务。这种方式充满了危险。相反，在写每一行时都三思而后行。可能会出现什么样的错误？是否已经考虑了所有可能出现的逻辑分支？放慢速度，有条不紊地编程虽然看上去很平凡，但这的确是减少缺陷的好办法。

一定要在完成与一个代码段相关的所有任务之后，再进入下一个环节。例如，如果决定先编写主体部分，再加入错误检查和处理，那么一定要确保这两项工作的完成都遵循章法。如果要推迟错误检查的编写，而直接开始编写超过 3 个代码段的主体部分，一定要慎之又慎。程序员也许真的想随后再回来编写错误检查，但却一而再再而三地向后推迟，这期间可能会忘记很多上下文，使得接下来的工作更加耗时和琐碎。

遵循章法是一种习惯，需要牢记于心并切实贯彻。如果不立即做正确的事，那么将来很可能也不会再去做正确的事。晚做不如早做，因为将来再做将需要遵循更多的章法。

4. 编译时打开所有警告开关

大多数语言的编译器都会给出一大堆错误信息。当这些编译器碰到潜在的有缺陷代码时（如在赋值之前使用 C 或 C++变量），它们也会给出各种各样的警告。通常情况下，这些警告可以有选择地启用或禁用。

如果代码中充满了危险的构造，将会得到数页的警告信息。糟糕的是，通常的反应是禁用编译器的警告功能，或者干脆不理会这些信息。这两种做法都不可取。

在任何情况下都要打开编译器的警告功能。如果代码产生了任何的警告信息，立即修正代码，让编译器的报错声停下来。在启用了警告功能之后，不要对不能安静地完成编译的代码感到满意。警告的出现总是有原因的。即使某个警告无关紧要，也不要置之不理。否则，总有一天这个警告会隐藏一个确实重要的警告。

5. 避免让任何人做不该做的修补工作

内部的事情就应该留在内部。私人的东西就应该用锁和钥匙保管起来。不要把代码初稿示于众人。不管多么礼貌地恳求，只要稍不注意，别人就会篡改你的数据，然后自以为是地试着调用“仅用于执行”的例行程序。

- 在面向对象的语言中，通过将属性设为专用（private）来防止对内部类数据的访问。在 C++中，可以考虑使用 Cheshire cat/pimpl idiom。（见参考书目 Meyers 97）
- 在过程语言中，仍然可以使用面向对象（OO）的打包概念，将 private 数据打包在不透明的类型背后，并提供可以操作它们的定义良好的公共函数。

将所有变量保持在尽可能小的范围内。不到万不得已，不要声明全局变量。如果变量可以声明为函数内的局部变量，就不要在文件范围上声明。如果变量可以声明为循环体内的局部变量，就不要在函数范围上声明。

6. 编码要清晰

如果要从简洁（但是有可能让人困惑）的代码和清晰（但是有可能比较冗长）的代码中选择，一定要选那些看上去和预期相符合的代码，即使它不太优雅。例如，将复杂的代数运算拆分为一系列单独的语句，使逻辑更清晰。

用户编写的代码也许需要一位初级程序员来进行维护，如果他不能理解代码的逻辑，那



么他肯定会犯一些错误。复杂的结构或不常用的语言技巧可以证明程序员在运算符优先级方面渊博的知识，但是这些实际上会扼杀代码的可维护性。请保持代码简单。

不能维护的代码是不安全的。举一个极端的例子，过于复杂的表达式会使编译器生成错误的代码，许多编译器优化的错误就是因此而造成的。

简单就是一种美，不要让代码过于复杂。

7. 使用安全的数据结构

如果大家做不到，那么就安全地使用危险的数据结构。

最常见的安全隐患大概是由缓冲溢出引起的。缓冲溢出是由于不正确地使用固定大小的数据结构而造成的。如果代码在没有检查一个缓冲的大小之前就写入这个缓冲，那么写入的内容总是有可能会超过缓冲的末尾的。

避免由于这些隐患而受到攻击其实很简单：不要编写这样的糟糕代码，使用更安全的、不允许破坏程序的数据结构。也可以对不安全的数据类型系统地使用安全的操作。

8. 尽可能推迟一些声明变量

尽可能推迟一些声明变量，可以使变量的声明位置与使用它的位置尽量接近，从而防止它干扰代码的其他部分。这样做也使得使用变量的代码更加清晰。不再需要到处寻找变量的类型和初始化，在附近声明使这些都变得非常明显。

不要在多个地方重用同一个临时变量，即使每次使用都是在逻辑上相互分离的区域中进行的。变量重用会使以后对代码重新完善的工作变得异常复杂。每次都创建一个新的变量——编译器会解决任何有关效率的问题。

9. 使用静态分析工具

编辑器警告是对代码的一次有限的静态分析（即在程序运行之前执行的代码检查）的结果。

还有许多独立的静态分析工具可供使用，如用于 C 语言的 lint（以及更多新出的衍生工具）和用于.NET 汇编程序的 FxCop。程序员的日常编程工作，应该包括使用这些工具来检查自己的代码。它们会比编译器挑出更多的错误。

10. 检查所有的返回值

如果一个函数返回一个值，它这样做肯定是有理由的。检查这个返回值。如果返回值是一个错误代码，就必须辨别这个代码并处理所有的错误。不要让错误悄无声息地侵入程序；忍受错误会导致不可预知的行为。

这既适用于用户自定义的函数，也适用于标准库函数。大多数难以察觉的错误都是因为程序员没有检查返回值而出现的。不要忘记，某些函数会通过不同的机制（例如，标准 C 库的 errno）返回错误。不论何时，都要在适当的级别上捕获和处理相应的异常。

11. 谨慎处理内存

对于在执行期间所获取的任何资源，必须彻底释放。内存是这类资源最常提到的一个例子，但并不是唯一的。文件和线程锁也是大家必须小心使用的宝贵资源。做一个好的“管家”。

不要因为觉得操作系统会在程序退出时清除程序，就不注意关闭文件或释放内存。对于代码还会执行多长时间，是否会耗尽所有的文件句柄或占用所有的内存，其实用户一无所知。用户甚至不能肯定操作系统是否会完全释放自己的资源，有的操作系统就不是这样的。

有一个学派说："在确定你的程序可以运行之前，不要担心内存的释放；只有在能够确定之后再添加所有相关的释放操作。"这种观点大错特错，是一种荒谬而且危险的做法。它会使用户在使用内存时出现许许多多的错误；用户将不可避免地在某些地方忘记释放内存。

Java 和.NET 使用垃圾回收器来执行这些繁重的清洁工作，所以用户可以"忘记"释放资源。让它们进入工作状态，这样在运行时将会不时地进行清扫。这真是一种享受，不过，不要因此而对安全性抱有错误的想法。用户必须显式地终止对那些不再需要，或不会被自动清除的对象的引用；不要意外地保留对对象的引用。不太先进的垃圾回收器也很容易会被循环引用蒙蔽（例如，A 引用 B，B 又引用 A，除此之外没有对 A 和 B 的引用）。这会导致对象永远不会被清除，这是一种难以发现的内存泄漏形式。

12. 在声明位置初始化所有变量

如果用户初始化了每个变量，它们的用途就会是明确的。依靠像"如果我不初始化它，我就不关心初始值"的经验主义是不安全的。未初始化的值以后可能随时都会变成问题。

C 和 C++在这个问题上更加复杂。如果用户意外地使用了一个没有初始化的变量，那么程序在每次运行的时候都将得到不同的结果，这取决于当时内存中的垃圾信息是什么。在一个地方声明一个变量，随后再对它进行赋值，在这之后再使用它，这样会为错误打开一个窗口。如果赋值的语句被跳过，就会花费大量的时间来寻找程序随机出现各种行为的原因。在声明每个变量的时候就对它进行初始化，就可以把这个窗口关上，因为即使初始化时赋的值是错误的，至少出现的错误行为也是可以预知的。

比较安全的语言（如 Java 和 C#）通过为所有变量定义初始值，回避了这个易犯的错误。在声明变量的时候对它进行初始化仍然是一种好的做法，这样可以提高代码的明确性。

13. 使用标准语言工具

明确地定义正在使用的是哪个语言版本。除非项目要求（最好是有一个好的理由），否则不要将命运交给编译器，或者对该语言的任何非标准的扩展。如果该语言的某个领域还没有定义，就不要依赖所使用的特定编译器的行为（例如，不要依赖 C 编译器将 char 作为有符号的值对待，因为其他的编译器并不是这样的）。这样做会产生非常脆弱的代码。当更新了编译器之后，会发生什么？一位新的程序员加入到开发团队中，如果他不理解那些扩展，会发生什么？依赖于特定编译器的个别行为，将导致以后难以发现的错误。

14. 谨慎强制转换

大多数语言都允许将数据从一种类型强制转换（或转换）为另一种类型。这种操作有时比其他操作更成功。如果试着将一个 64 位的整数转换为较小的 8 位数据类型，那么其他的 56 位会怎么样呢？执行环境可能会突然抛出异常，或者悄悄地使数据的完整性降级。很多程序员并不考虑这类事情，所以他们的程序就会表现出不正常的行为。

如果真的想使用强制转换，就必须对之深思熟虑。程序员所告诉编译器的是："忘记类型检查吧，我知道这个变量是什么，而你并不知道。"在类型系统中撕开了一个大洞，并直接穿越过去。这样做很不可靠。如果犯了任何一种错误，编译器将只会静静地坐在那里小声嘀咕道："我告诉过你的。"如果很幸运（例如使用 Java 或 C#），运行时可能会抛出异常以让程序员了解发生了错误，但这完全依赖于要进行的是什么转换。



C 和 C++对于数据类型的精度并不明确，所以对于数据类型的可互换性不要做任何假设。不要假设 int 和 long 的大小相同并且可以相互赋值，即使在平台上侥幸可以这样做。代码可以在平台之间移植，但是糟糕的代码可移植性很差。

15. 使用好的诊断信息日志工具

当程序员编写新的代码时，常常会加入很多诊断信息，以确定程序的运行情况。在调试结束后是否应该删除这些诊断信息呢？保留这些信息对以后再次访问代码会带来很多方便，特别是如果在此期间可以有选择地禁用这些信息。

有很多诊断信息日志系统可以帮助实现这种功能。这些系统中很多都可以使诊断信息在不需要的时候不带来任何开销；可以有选择地使它们不参加编译。

13.4　性能测试

在开发中，性能测试是设计初期容易忽略的问题，开发人员会为了解决一个问题而“不择手段”，字符串拼接、大量的网络调用和数据库访问等都对系统的性能产生了影响，可是大家不会关心这些问题，只会做出“CPU 速度在变快”，“内存在变大”，并且“好像也没有那么慢吧”的回答。

当然，当前有很多商业的性能测试软件可供使用，如 JProfiler 和 JProbe Profiler 等，但在开发当中显得有些遥远而又昂贵。

13.4.1　背景

面向对象编程通过抽象继承采用模块化的思想来求解问题域，但是模块化不能很好地解决所有问题。有时，这些问题可能在多个模块中都出现，像日志功能，为了记录每个方法进入和离开时的信息，程序员不得不在每个方法里添加 log("in some method")等信息。如何解决这类问题呢？将这些解决问题的功能点散落在多个模块中会使冗余增大，并且当很多个功能点出现在一个模块中时，代码变得很难维护。因此，AOP（Aspect Oriented Programming）应运而生。如果说 OOP（Aobject Oriented Programming）关注的是一个类的垂直结构，那么 AOP 是从水平角度来看待问题。

动态代理类可以在运行时实现若干接口，每一个动态代理类都有一个 InvocationHandler 对象与之对应，这个对象实现了 InvocationHandler 接口，通过动态代理的接口对动态代理对象的方法调用会转而调用 InvocationHandler 对象的 invoke 方法，通过动态代理实例、方法对象和参数对象可以执行调用并返回结果。

每当提起 AOP，大家首先会想到的是日志记录、权限检查和事务管理。AOP 是解决这些问题的好办法。本文根据 AOP 的思想，通过动态代理来解决一类新的问题——性能测试（performance testing）。

性能测试主要包括以下几个方面。

- 计算性能：可能是人们首先关心的，简单地说就是执行一段代码所用的时间。
- 内存消耗：程序运行所占用的内存大小。
- 启动时间：从启动程序到程序正常运行的时间。
- 可伸缩性(Scalability)。

- 用户察觉性能(Perceived Performance)：不是程序实际运行有多快,而是用户感觉程序运行有多快。

13.4.2 Java代理

简单说，代理就是在客户和目标对象之间设立一个中介，就好比房屋建筑商和买家之间的关系，买家不可能直接到房屋建筑商那里去买房，在买家和建家之间往往都存在一个售房机构，这个售房机构的作用其实就是一个代理，他在买家和建家之间建立了一个交流渠道。买家通过售房机构向建筑商买房。

为什么要存在代理呢？还拿房屋买卖这个例子来解释为什么要存在代理，其实，在房屋买卖里只有两个主要实体，一个就是房屋的建筑商，另一个就是购买者，那为什么还要在二者之间建立一个中介呢？房屋建筑者那里肯定有一些东西是不希望让买房者知道的，比如说成本。还有一个原因就是，房屋建筑商他的主要精力应当放在房屋建造上，因此，他需要设立一个代理商来为他代理房屋出售，Java 的代理模式所起的功能与之大同小异。

Java 代理主要分为两类：静态代理和动态代理。

（1）静态代理

静态代理可以代理抽象类和接口，举例说明。

1）接口（抽象类）。

```
public interface SayHellow{
  public void say();
}
```

2）接口实现类。

```
public class SayHellowImpl implements sayHellow {
  public void say(){
    System.out.println("HelloWorld");
  }
}
```

3）静态代理类。

```
public class StaticProxy implements SayHellow {
  private SayHellow sayHellow;
  public StaticProxy(SayHellow sayHellow){
    this.sayHellow=sayHellow;
  }
  public void say(){
    System.out.println("Say to world:");
    sayHellow.say();
  }
}
```

该代理的弊端是，如果接口有变动，比如说在里面加入一个新的方法。那么代理类也要



进行更改。代码的耦合度比较高，不利于程序的扩展。

（2）动态代理

只能代理接口。所有的动态代理必须实现 Java 提供的一个借口 InvocationHandler，重写它的 invoke 方法，该 invoke 方法就是调用被代理接口的所有方法时需要调用的，该 invoke 方法返回的值是被代理接口的一个实现类。

```
public class DynamicProxy implements InvocationHandler{
private Object object;
    //绑定关系，也就是关联到哪个接口（与具体的实现类绑定）的哪些方法将被调用时，执行 invoke 方法
    //Proxy.newProxyInstance 的第三个参数是表明这些被拦截的方法执行时需要执行哪个 InvocationHandler 的 invoke 方法
     public Object bindRelation(Object object){
    this.object = object;
    return Proxy.newProxyInstance(object.getClass().getClassLoader(), object.getClass().getInterfaces(),this);
       }
     //拦截关联的这个实现类的方法被调用时将被执行
public Object invoke(Object proxy, Method method, Object[] args) throws Throwable {
        System.out.println("Welcome");
  Object result = method.invoke(object, args);
  return result;
  }

}
```

一段测试类代码如下。

```
public class DynamicProxyTest {
  public static void main(String[] args) {
    SayHellow sayHellow = new SayHelloImpl();
    DynamicProxy dp = new DynamicProxy();
    SayHellow helloWorld1 = (SayHellow)dp.bindRelation(sayHellow);
    helloWorld1.print();
  }
}
```

13.4.3　性能测试的步骤

在每种不同的系统架构的实施中，开发人员可能选择不同的实现方式，造成实际情况纷繁复杂。不可能对每种技术都详细解说，这里只是介绍一种方法提供给大家如何选择测试策略，从而帮助分析软件不同部分的性能指标，进而分析出整体架构的性能指标和性能瓶颈。

因为工程和项目的不同，所选用的度量，评估方法也有不同之处。不过仍然有一些通用的步骤帮助大家完成一个性能测试项目。性能测试的一般步骤如下。

1）制定目标和分析系统。

2）选择测试度量的方法。

3）学习的相关技术和工具。

4）制定评估标准。

5）设计测试用例。

6）运行测试用例。

7）分析测试结果。

13.4.4 计算机性能测试

在 Java 中为用户提供了 System.currentTimeMillis()方法，可以得到毫秒级的当前时间，在以前的程序当中一定也写过类似的代码来计算执行某一段代码所消耗的时间。使用格式如下。

```
long start=System.currentTimeMillis();
doSth();
long end=System.currentTimeMillis();
System.out.println("time lasts "+(end-start)+"ms");
```

但是，如果在每个方法里面都写上这么一段代码，是一件很枯燥烦琐的事情，其实可以通过 Java 的 java.lang.reflect.Proxy 和 java.lang.reflect.InvocationHandler 利用动态代理来很好地解决上面的问题。

本节将通过一个实例的实现过程，来讲解计算机性能测试的方法。本实例要测试的是 java.util.LinkedList 方法和 java.util.ArrayList 的 get(int index)方法，ArrayList 要比 LinkedList 高效，因为前者是随机访问，而后者需要顺序访问。

本实例代码保存在“光盘:\daima\13\”中，命名为 TestComputer。下面开始介绍本实例的具体实现过程。

第 1 步：编写主布局文件 main.xml，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

第 2 步：编写文件 Testing.java，用于创建一个接口。具体代码如下所示。

```
package com.TestComputer;
public interface Testing
{
    public  void  testArrayList();
```



```
        public    void    testLinkedList();
}
```

第 3 步：编写文件 TestComputer.java，用于创建测试对象来实现这个接口。具体代码如下所示。

```
package com.TestComputer;
import java.util.ArrayList;
import java.util.LinkedList;
import java.util.List;
public class TestComputer implements Testing
{
    private List link   = new LinkedList();
    private List array = new ArrayList();
    public TestComputer()
    {
        for (int i = 0; i < 10000; i++)
        {
            array.add(new Integer(i));
            link.add(new Integer(i));
        }
    }
    public void testArrayList()
    {
        for (int i = 0; i < 10000; i++)
            array.get(i);
    }
    public void testLinkedList()
    {
        for (int i = 0; i < 10000; i++)
            link.get(i);
    }
}
```

第 4 步：编写文件 Handler.java，用于实现 InvocationHandler 接口。具体代码如下所示。

```
package com.TestComputer;
import java.lang.reflect.InvocationHandler;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;
import android.util.Log;
public class Handler implements InvocationHandler
{
    private Object      obj;
    public Handler(Object obj)
    {
```

```
            this.obj = obj;
        }
        public static Object newInstance(Object obj)
        {
            Object result = Proxy.newProxyInstance(obj.getClass().getClassLoader(), obj.getClass().
    getInterfaces(), new Handler(obj));
            return (result);
        }
        public Object invoke(Object proxy, Method method, Object[] args) throws Throwable
        {
            Object result;
            try
            {
                Log.i("Handler", "begin method " + method.getName());

                long start = System.currentTimeMillis();
                result = method.invoke(obj, args);
                long end = System.currentTimeMillis();

                Log.i("Handler", "the method " + method.getName() + " lasts " + (end - start) + "ms");
            }
            catch (InvocationTargetException e)
            {
                throw e.getTargetException();
            }
            catch (Exception e)
            {
                throw new RuntimeException("unexpected invocation exception: " + e.getMessage());
            }
            finally
            {
                Log.i("Handler", "end method " + method.getName());
            }
            return result;
        }
    }
```

第 5 步：编写文件 Activity01.java，功能是加入测试程序。具体代码如下所示。

```
    package com.TestComputer;
    import com.TestComputer.R;

    import android.app.Activity;
    import android.os.Bundle;

    public class Activity01 extends Activity
    {
```



```
        /** Called when the activity is first created. */
        @Override
        public void onCreate(Bundle savedInstanceState)
        {
            super.onCreate(savedInstanceState);

            try
            {
                Testing  testing  =  (Testing)  Handler.newInstance(new  TestComputer());
                testing.testArrayList();
                testing.testLinkedList();
            }catch  (Exception  e){
                e.printStackTrace();
            }
            setContentView(R.layout.main);
        }
    }
```

至此整个实例讲解完毕，执行后的效果如图 13-1 所示。

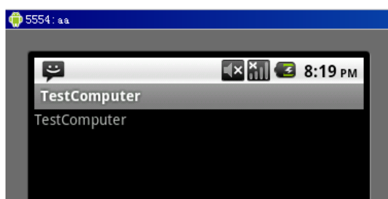


图 13-1　执行效果

在 DDMS 中可以看到详细的 Log 信息，可以看出每个方法的执行时间，如图 13-2 所示。

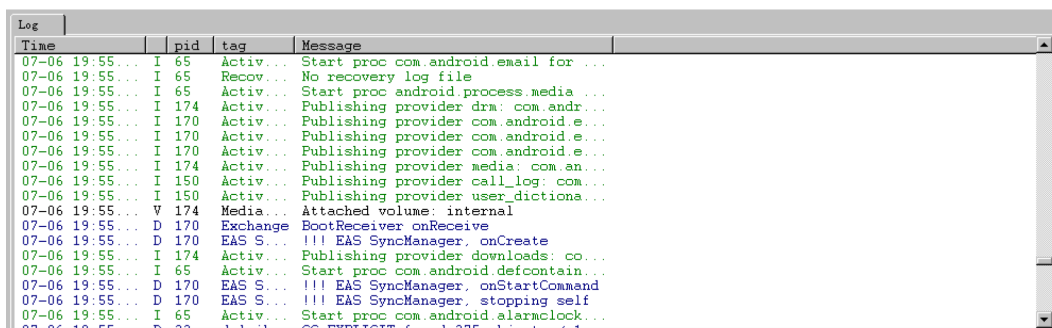


图 13-2　Log 信息

通过上述实例，演示了利用动态代理比较两个方法的执行时间，有时候通过一次简单的测试进行比较是片面的，读者可以进行多次执行测试对象，从而计算出最差、最好和平均性能。这样，才能加快经常执行的程序的速度，尽量少调用速度慢的程序。

使用动态代理的好处是不必修改原有代码 FooImpl，但是缺点是不得不写一个接口，如果类原来没有实现接口的话。

13.4.5　内存性能测试

当一个 Java 应用程序运行时，有很多需要消耗内存的因素存在，像对象、加载类、线程等。在这里只考虑程序中的对象所消耗的虚拟机堆空间，这样就可以利用 Runtime 类的 freeMemory()方法和 totalMemory()方法。

在本节的内容中，将通过一个具体实例的实现过程，来讲解内存性能测试的方法。本实例代码保存在“光盘:\daima\13\”中，命名为 Testmemory。下面开始介绍本实例的具体实现过程。

第 1 步：编写主布局文件 main.xml，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/hello"
    />
</LinearLayout>
```

第 2 步：编写文件 AA.java，在此创建一个接口 AA。具体代码如下所示。

```
package com.Testmemory;
public interface AA
{
     public void creatArray();
     public void creatHashMap();
}
```

第 3 步：编写文件 BB.java，用于创建对象实现接口 BB。其功能是分别比较一个长度为 1000 的 ArrayList 和 HashMap 所占内存空间的大小。具体实现代码如下所示。

```
package com.Testmemory;
import java.util.ArrayList;
import java.util.HashMap;
public class BB implements AA
{
     ArrayList        arr    = null;
     HashMap          hash   = null;
     public void creatArray()
     {
          arr = new ArrayList(1000);
     }
     public void creatHashMap()
```



```
            {
                hash = new HashMap(1000);
            }
        }
```

第 4 步：编写文件 Memory.java，在此定义一个类，用于计算当前的内存消耗。具体代码如下所示。

```
package com.Testmemory;

public class Memory
{
    public static long used()
    {
        long total = Runtime.getRuntime().totalMemory();
        long free = Runtime.getRuntime().freeMemory();
        return (total - free);
    }
}
```

第 5 步：编写文件 Handle.java 用于修改 Handle 类的 invoke()方法。具体实现代码如下所示。

```
package com.Testmemory;

import java.lang.reflect.InvocationHandler;
import java.lang.reflect.InvocationTargetException;
import java.lang.reflect.Method;
import java.lang.reflect.Proxy;

import android.util.Log;

public class Handler implements InvocationHandler
{
    private Object      obj;
    public Handler(Object obj)
    {
        this.obj = obj;
    }
    public static Object newInstance(Object obj)
    {
        Object result = Proxy.newProxyInstance(obj.getClass().getClassLoader(), obj.getClass().
getInterfaces(), new Handler(obj));
        return (result);
    }
    public Object invoke(Object proxy, Method method, Object[] args) throws Throwable
    {
        Object result;
```

```
                try
                {
                    Log.i("Handler", "begin method " + method.getName());

                    long start = Memory.used();
                    result = method.invoke(obj, args);
                    long end = Memory.used();

                    Log.i("Handler", "memory increased by " + (end - start) + "bytes");
                }
                catch (InvocationTargetException e)
                {
                    throw e.getTargetException();
                }
                catch (Exception e)
                {
                    throw new RuntimeException("unexpected invocation exception: " + e.getMessage());
                }
                finally
                {
                    Log.i("Handler", "end method " + method.getName());
                }
                return result;
            }
        }
```

至此，整个实例讲解完毕，执行后的效果如图 13-3 所示。

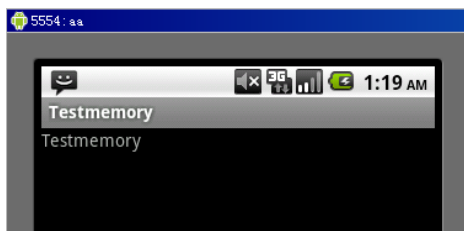


图 13-3　执行效果

在 DDMS 中可以看到详细的 Log 信息，可以看出所占用的内存空间，如图 13-4 所示。

Log

Time		pid	tag	Message
07-06 19:54...	D	33	dalvikvm	GC_FOR_MALLOC freed 10024 objects...
07-06 19:54...	D	33	dalvikvm	GC_FOR_MALLOC freed 8084 objects ...
07-06 19:54...	D	33	dalvikvm	GC_FOR_MALLOC freed 7906 objects ...
07-06 19:54...	D	33	dalvikvm	GC_FOR_MALLOC freed 7645 objects ...
07-06 19:54...	D	33	dalvikvm	GC_FOR_MALLOC freed 7674 objects ...
07-06 19:54...	D	33	dalvikvm	GC_EXPLICIT freed 6657 objects / ...
07-06 19:54...	D	33	dalvikvm	GC_EXPLICIT freed 1539 objects / ...
07-06 19:54...	D	33	dalvikvm	GC_EXPLICIT freed 447 objects / 2...
07-06 19:54...	D	33	dalvikvm	GC_EXPLICIT freed 315 objects / 2...
07-06 19:54...	I	33	Zygote	...preloaded 1265 classes in 2216...
07-06 19:54...	E	33	Zygote	setreuid() failed. errno: 17
07-06 19:54...	D	33	dalvikvm	GC_EXPLICIT freed 104 objects / 1...
07-06 19:54...	I	33	Zygote	Preloading resources...
07-06 19:54...	W	33	Zygote	Preloaded drawable resource #0x10...
07-06 19:54...	W	33	Zygote	Preloaded drawable resource #0x10...
07-06 19:54...	W	33	Zygote	Preloaded drawable resource #0x10...
07-06 19:54...	W	33	Zygote	Preloaded drawable resource #0x10...
07-06 19:54...	W	33	Zygote	Preloaded drawable resource #0x10...

图 13-4　Log 信息



AOP 通过分解关注点和 OOP 相互结合，使程序更加简洁易懂，通过 Java 语言本身提供的动态代理帮助很容易分解关注点，取得了较好的效果。不过测试对象必须实现接口在一定程度上限制了动态代理的使用，可以借鉴 Spring 中使用的 CGlib 来为没有实现任何接口的类创建动态代理。

注意：其实在本实例中，笔者没有遵循命名规范，只是随便起了 AA 和 BB，不知细心的读者发现了吗？

13.5　程序优化

对于任何一个程序来说，可供利用的内存、CPU 和带宽都是有限的。使用优化的目的就是让程序尽量减少对这些资源的占有。本节将详细阐述程序优化的基本知识。

13.5.1　Java程序的基本优化

在 Java 程序中，性能问题的大部分原因并不在于 Java 语言，而是在于程序本身。养成好的代码编写习惯非常重要，比如正确地、巧妙地运用 java.lang.String 类和 java.util.Vector 类，它能够显著地提高程序的性能。下面就来具体地分析这方面的问题。

1）尽量指定类的 final 修饰符，带有 final 修饰符的类是不可派生的。在 Java 核心 API 中，有许多应用 final 的例子，如 java.lang.String。为 String 类指定 final 防止了人们覆盖 length() 方法。另外，如果指定一个类为 final，则该类所有的方法都是 final。Java 编译器会寻找机会内联（inline）所有的 final 方法（这和具体的编译器实现有关）。此举能够使性能平均提高 50% 。

2）尽量重用对象。特别是 String 对象的使用中，出现字符串连接情况时应用 StringBuffer 代替。由于系统不仅要花时间生成对象，以后可能还需花时间对这些对象进行垃圾回收和处理。因此，生成过多的对象将会给程序的性能带来很大的影响。

3）尽量使用局部变量，调用方法时传递的参数以及在调用中创建的临时变量都保存在栈（Stack）中，速度较快。其他变量，如静态变量、实例变量等，都在堆（Heap）中创建，速度较慢。另外，依赖于具体的编译器/JVM，局部变量还可能得到进一步优化。请参见《尽可能使用堆栈变量》。

4）不要重复初始化变量。默认情况下，调用类的构造函数时， Java 会把变量初始化成确定的值：所有的对象被设置成 null，整数变量（byte、short、int、long）设置成 0，float 和 double 变量设置成 0.0，逻辑值设置成 false。当一个类从另一个类派生时，这一点尤其应该注意，因为用 new 关键词创建一个对象时，构造函数链中的所有构造函数都会被自动调用。

5）在 Java + Oracle 的应用系统开发中，Java 中内嵌的 SQL 语句尽量使用大写的形式，以减轻 Oracle 解析器的解析负担。

6）Java 编程过程中，进行数据库连接、I/O 流操作时务必小心，在使用完毕后，即时关闭以释放资源。因为对这些大对象的操作会造成系统大的开销，稍有不慎，会导致严重的后果。

7）由于 JVM 有其自身的 GC 机制，不需要程序开发者的过多考虑，从一定程度上减轻了开发者负担，但同时也遗漏了隐患，过分地创建对象会消耗系统的大量内存，严重时会导

致内存泄露，因此，保证过期对象的及时回收具有重要意义。JVM 回收垃圾的条件是：对象不在被引用；然而，JVM 的 GC 并非十分的机智，即使对象满足了垃圾回收的条件也不一定会被立即回收。所以，建议在对象使用完毕手动置成 null。

8）在使用同步机制时，应尽量使用方法同步代替代码块同步。

9）尽量减少对变量的重复计算，例如：

```
for(int i = 0;i < list.size; i ++) {
        …
}
```

应替换为

```
for(int i = 0,int len = list.size();i < len; i ++) {
        …
}
```

10）尽量采用 lazy loading 的策略，即在需要的时候才开始创建。例如：

```
    String str = “aaa”;
        if(i == 1) {
            list.add(str);
}
```

应替换为

```
        if(i == 1) {
String str = “aaa”;
            list.add(str);
}
```

11）慎用异常，异常对性能不利。抛出异常首先要创建一个新的对象。Throwable 接口的构造函数调用名为 fillInStackTrace()的本地（Native）方法，fillInStackTrace()方法检查堆栈，收集调用跟踪信息。只要有异常被抛出，VM 就必须调整调用堆栈，因为在处理过程中创建了一个新的对象。异常只能用于错误处理，不应该用来控制程序流程。

12）不要在循环中使用：

```
Try {
} catch() {
}
```

应把其放置在最外层。

13）注意 StringBuffer 的使用，StringBuffer 表示了可变的、可写的字符串。有如下 3 个构造方法。

```
StringBuffer ();                //默认分配 16 个字符的空间
StringBuffer (int size);        //分配 size 个字符的空间
StringBuffer (String str);      //分配 16 个字符+str.length()个字符空间
```



可以通过 StringBuffer 的构造函数来设定它的初始化容量，这样可以明显地提升性能。这里提到的构造函数是 StringBuffer(int length)，length 参数表示当前的 StringBuffer 能保持的字符数量。也可以使用 ensureCapacity(int minimumcapacity)方法在 StringBuffer 对象创建之后设置它的容量。首先看看 StringBuffer 的默认行为，然后再找出一条更好的提升性能的途径。

StringBuffer 在内部维护一个字符数组，当使用默认的构造函数来创建 StringBuffer 对象的时候，因为没有设置初始化字符长度，StringBuffer 的容量被初始化为 16 个字符，也就是说默认容量就是 16 个字符。当 StringBuffer 达到最大容量的时候，它会将自身容量增加到当前的 2 倍再加 2，也就是（2×旧值+2）。如果使用默认值，初始化之后接着往里面追加字符，在追加到第 16 个字符的时候它会将容量增加到 34（2×16+2），当追加到 34 个字符的时候就会将容量增加到 70（2×34+2）。无论何事只要 StringBuffer 到达它的最大容量它就不得不创建一个新的字符数组然后重新将旧字符和新字符都复制一遍——这也太昂贵了点。所以总是给 StringBuffer 设置一个合理的初始化容量值是错不了的，这样会带来立竿见影的性能增益。

StringBuffer 初始化过程的调整的作用由此可见一斑。所以，使用一个合适的容量值来初始化 StringBuffer 永远都是一个最佳的建议。

14）合理地使用 Java 类 java.util.Vector。

简单地说，一个 Vector 就是一个 java.lang.Object 实例的数组。Vector 与数组相似，它的元素可以通过整数形式的索引访问。但是，Vector 类型的对象在创建之后，对象的大小能够根据元素的增加或者删除而扩展、缩小。看下面这段向 Vector 加入元素的代码。

```
Object obj = new Object();
Vector v = new Vector(100000);
for(int I=0;
I<100000; I++) { v.add(0,obj); }
```

除非有绝对充足的理由要求每次都把新元素插入到 Vector 的前面，否则上面的代码对性能不利。在默认构造函数中，Vector 的初始存储能力是 10 个元素，如果新元素加入时存储能力不足，则以后存储能力每次加倍。Vector 类就像 StringBuffer 类一样，每次扩展存储能力时，所有现有的元素都要复制到新的存储空间之中。下面的代码片段要比前面的例子快几个数量级。

```
Object obj = new Object();
Vector v = new Vector(100000);
for(int I=0; I<100000; I++) { v.add(obj); }
```

同样的规则也适用于 Vector 类的 remove()方法。由于 Vector 中各个元素之间不能含有“空隙”，删除除最后一个元素之外的任意其他元素都导致被删除元素之后的元素向前移动。也就是说，从 Vector 删除最后一个元素要比删除第一个元素“开销”低好几倍。

假设要从前面的 Vector 删除所有元素，可以使用以下代码。

```
for(int I=0; I<100000; I++){
v.remove(0);
```

```
}
```

但是，与下面的代码相比，前面的代码要慢几个数量级。

```
for(int I=0; I<100000; I++) {
v.remove(v.size()-1);
}
```

从 Vector 类型的对象 v 删除所有元素的最好方法如下。

```
v.removeAllElements();
```

假设 Vector 类型的对象 v 包含字符串“Hello”。考虑下面的代码，它要从这个 Vector 中删除“Hello”字符串。

```
String s = "Hello";
int i = v.indexOf(s);
if(I != -1) v.remove(s);
```

这些代码看起来没什么错误，但它同样对性能不利。在这段代码中，indexOf()方法对 v 进行顺序搜索寻找字符串 “Hello”，remove(s)方法也要进行同样的顺序搜索。改进之后的代码如下。

```
String s = "Hello";
int i = v.indexOf(s);
if(I != -1) v.remove(i);
```

在上述代码中直接在 remove()方法中给出待删除元素的精确索引位置，从而避免了第二次搜索。一个更好的代码是

```
String s = "Hello"; v.remove(s);
```

最后，再来看一个有关 Vector 类的代码片段。

```
for(int I=0; I++;I < v.length)
```

如果 v 包含 100 000 个元素，这个代码片段将调用 v.size()方法 100 000 次。虽然 size 方法是一个简单的方法，但它仍旧需要一次方法调用的开销，至少 JVM 需要为它配置以及清除堆栈环境。在这里，for 循环内部的代码不会以任何方式修改 Vector 类型对象 v 的大小，因此上面的代码最好改写成下面这种形式。

```
int size = v.size(); for(int I=0; I++;I<size)
```

虽然这是一个简单的改动，但它仍旧赢得了性能。毕竟，每一个 CPU 周期都是宝贵的。

15）当复制大量数据时，使用 System.arraycopy()命令。

16）使用代码重构，增强代码的可读性。例如：

```
public class ShopCart {
```



```
                private List carts ;
                …
                public void add (Object item) {
                        if(carts == null) {
                                carts = new ArrayList();
        }
        crts.add(item);
        }
        public void remove(Object item) {
                if(carts. contains(item)) {
                        carts.remove(item);
        }
        }
        public List getCarts() {
                //返回只读列表
                return Collections.unmodifiableList(carts);
        }

        //不推荐这种方式
        //this.getCarts().add(item);
        }
```

17）不用 new 关键词创建类的实例。用 new 关键词创建类的实例时，构造函数链中的所有构造函数都会被自动调用。但如果一个对象实现了 Cloneable 接口，可以调用它的 clone()方法。clone()方法不会调用任何类构造函数。

在使用设计模式（Design Pattern）的场合，如果用 Factory 模式创建对象，则改用 clone()方法创建新的对象实例非常简单。例如，下面是 Factory 模式的一段典型代码。

```
public static Credit getNewCredit() {
        return new Credit();
}
```

改进后的代码使用 clone()方法，如下所示。

```
private static Credit BaseCredit = new Credit();
public static Credit getNewCredit() {
        return (Credit) BaseCredit.clone();
}
```

上述思路对于数组处理同样很有用。

18）谨慎乘法和除法，看下面的代码：

```
for (val = 0; val < 100000; val +=5) {
alterX = val * 8; myResult = val * 2;
}
```

用移位操作替代乘法操作可以极大地提高性能。下面是修改后的代码。

```
for (val = 0; val < 100000; val += 5) {
alterX = val << 3; myResult = val << 1;
}
```

修改后的代码不再做乘以 8 的操作，而是改用等价的左移 3 位操作，每左移 1 位相当于乘以 2。相应地，右移 1 位操作相当于除以 2。值得一提的是，虽然移位操作速度快，但可能使代码比较难于理解，所以最好加上一些注释。

19）在 JSP 页面中关闭无用的会话。

一个常见的误解是以为 session 在有客户端访问时就被创建，然而事实是直到某 server 端程序调用 HttpServletRequest.getSession(true)这样的语句时才被创建，注意如果 JSP 没有显式地使用 <%@page session="false"%> 关闭 session，则 JSP 文件在编译成 Servlet 时将会自动加上这样一条语句 HttpSession session = HttpServletRequest.getSession(true);这也是 JSP 中隐含的 session 对象的来历。由于 session 会消耗内存资源，因此，如果不打算使用 session，应该在所有的 JSP 中关闭它。

对于那些无须跟踪会话状态的页面，关闭自动创建的会话可以节省一些资源。使用如下 page 指令。

```
<%@ page session="false"%>
```

20）JDBC 与 I/O。如果应用程序需要访问一个规模很大的数据集，则应当考虑使用块提取方式。默认情况下，JDBC 每次提取 32 行数据。举例来说，假设要遍历一个 5000 行的记录集，JDBC 必须调用数据库 157 次才能提取到全部数据。如果把块大小改成 512，则调用数据库的次数将减少到 10 次。

21）Servlet 与内存使用。许多开发者随意地把大量信息保存到用户会话之中。一些时候，保存在会话中的对象没有及时地被垃圾回收机制回收。从性能上看，典型的症状是用户感到系统周期性地变慢，却又不能把原因归于任何一个具体的组件。如果监视 JVM 的堆空间，它的表现是内存占用不正常的大起大落。

解决这类内存问题主要有 2 种办法。第一种办法是，在所有作用范围为会话的 Bean 中实现 HttpSessionBindingListener 接口。这样，只要实现 valueUnbound()方法，就可以显式地释放 Bean 使用的资源。另外一种办法就是尽快地把会话作废。大多数应用服务器都有设置会话作废间隔时间的选项。另外，也可以用编程的方式调用会话的 setMaxInactiveInterval()方法，该方法用来设定在作废会话之前，Servlet 容器允许的客户请求的最大间隔时间，以秒计。

22）使用缓冲标记。一些应用服务器加入了面向 JSP 的缓冲标记功能。例如，BEA 的 WebLogic Server 从 6.0 版本开始支持这个功能，Open Symphony 工程也同样支持这个功能。JSP 缓冲标记既能够缓冲页面片断，也能够缓冲整个页面。当 JSP 页面执行时，如果目标片断已经在缓冲之中，则生成该片断的代码就不用再执行。页面级缓冲捕获指定 URL 的请求，并缓冲整个结果页面。对于购物篮、目录以及门户网站的主页来说，这个功能极其有用。对于这类应用，页面级缓冲能够保存页面执行的结果，供后继请求使用。

23）选择合适的引用机制。

在典型的 JSP 应用系统中，页头、页脚部分往往被抽取出来，然后根据需要引入页头、



页脚。当前，在 JSP 页面中引入外部资源的方法主要有两种：include 指令、include 动作。

include 指令：例如<%@ include file="copyright.html" %>。该指令在编译时引入指定的资源。在编译之前，带有 include 指令的页面和指定的资源被合并成一个文件。被引用的外部资源在编译时就确定，比运行时才确定资源更高效。

include 动作：例如<jsp:include page="copyright.jsp" />。该动作引入指定页面执行后生成的结果。由于它在运行时完成，因此对输出结果的控制更加灵活。但是，只有当被引用的内容频繁地改变时，或者在对主页面的请求没有出现之前，被引用的页面无法确定时，使用 include 动作才合算。

24）及时清除不再需要的会话。

为了清除不再活动的会话，许多应用服务器都有默认的会话超时时间，一般为 30min。当应用服务器需要保存更多会话时，如果内存容量不足，操作系统会把部分内存数据转移到磁盘，应用服务器也可能根据“最近最频繁使用”（Most Recently Used）算法把部分不活跃的会话转储到磁盘，甚至可能抛出“内存不足”异常。在大规模系统中，串行化会话的代价是很昂贵的。当会话不再需要时，应当及时调用 HttpSession.invalidate()方法清除会话。HttpSession.invalidate()方法通常可以在应用的退出页面调用。

25）不要将数组声明为：public static final。

26）HashMap 的遍历效率讨论。

经常遇到对 HashMap 中的 key 和 value 值对的遍历操作，有如下两种方法。

```
Map<String, String[]> paraMap = new HashMap<String, String[]>();
//第一个循环
Set<String> appFieldDefIds = paraMap.keySet();
for (String appFieldDefId : appFieldDefIds) {
String[] values = paraMap.get(appFieldDefId);
...
}
//第二个循环
for(Entry<String, String[]> entry : paraMap.entrySet()){
String appFieldDefId = entry.getKey();
String[] values = entry.getValue();
...
}
```

上述第一种实现效率明显不如第二种实现的。

Set<String> appFieldDefIds = paraMap.keySet();是先从 HashMap 中取得 keySet 的，具体代码如下。

```
public Set<K> keySet() {
Set<K> ks = keySet;
return (ks != null ? ks : (keySet = new KeySet()));
}

private class KeySet extends AbstractSet<K> {
```

```
public Iterator<K> iterator() {
return newKeyIterator();
}
public int size() {
return size;
}
public boolean contains(Object o) {
return containsKey(o);
}
public boolean remove(Object o) {
return HashMap.this.removeEntryForKey(o) != null;
}
public void clear() {
HashMap.this.clear();
}
}
```

这其实是返回一个私有类 KeySet, 它是从 AbstractSet 继承而来，实现了 Set 接口。接下来看 for/in 循环的语法。

```
for(declaration : expression_r)
statement
```

在执行阶段被翻译成如下格式。

```
for(Iterator<E> #i = (expression_r).iterator(); #i.hashNext();){
declaration = #i.next();
statement
}
```

所以在第一个 for 语句 for (String appFieldDefId:appFieldDefIds) 中调用了 HashMap.keySet().iterator()。而这个方法调用了 newKeyIterator()，代码如下。

```
Iterator<K> newKeyIterator() {
return new KeyIterator();
}
private class KeyIterator extends HashIterator<K> {
public K next() {
return nextEntry().getKey();
}
}
```

在第二个循环 for(Entry<String, String[]> entry : paraMap.entrySet())中使用的 Iterator，是如下的一个内部类。

```
private class EntryIterator extends HashIterator<Map.Entry<K,V>> {
public Map.Entry<K,V> next() {
return nextEntry();
```



```
        }
    }
```

此时第一个循环得到 key，第二个循环得到 HashMap 的 Entry。效率就是从循环里面体现出来的，第二个循环可以直接取 key 和 value 值，而第一个循环还是得再利用 HashMap 的 get(Object key)来取 value 值。

接下来看 HashMap 的 get(Object key)方法，具体代码如下。

```
public V get(Object key) {
Object k = maskNull(key);
int hash = hash(k);
int i = indexFor(hash, table.length); //Entry[] table
Entry<K,V> e = table;
while (true) {
if (e == null)
return null;
if (e.hash == hash && eq(k, e.key))
return e.value;
        e = e.next;
}
}
```

其实就是再次利用 Hash 值取出相应的 Entry 做比较得到结果，所以使用第一种循环相当于两次进入 HashMap 的 Entry 中。而第二个循环取得 Entry 的值之后直接取 key 和 value，效率比第一个循环高。其实按照 Map 的概念来看，也应该是用第二个循环好一点，它本来就是 key 和 value 的值对，将 key 和 value 分开操作在这里不是好的选择。

27）array(数组)和 ArryList 的使用，先看两者特点。

- array（[]）：最高效，但是其容量固定且无法动态改变。
- ArrayList：容量可动态增长，但牺牲效率。

基于效率和类型检验，应尽可能使用 Array，无法确定数组大小时才使用 ArrayList。ArrayList 是 Array 的复杂版本。

在 ArrayList 内部封装了一个 Object 类型的数组，从一般意义来说，它和数组没有本质的差别，甚至于 ArrayList 的许多方法，如 Index、IndexOf、Contains、Sort 等都是在内部数组的基础上直接调用 Array 的对应方法。

当 ArrayList 存入对象时，抛弃类型信息，所有对象屏蔽为 Object，编译时不检查类型，但是运行时会报错。

注意：从 jdk5 中加入了对泛型的支持，已经可以在使用 ArrayList 时进行类型检查。

由此可见，ArrayList 与数组的区别主要就是由于动态增容的效率问题了。

28）尽量使用 HashMap 和 ArrayList，除非必要，否则不推荐使用 HashTable 和 Vector，后者由于使用同步机制，而导致了性能的开销。

29）StringBuffer 和 StringBuilder 的区别。

java.lang.StringBuffer 线程安全的可变字符序列。一个类似于 String 的字符串缓冲区，但

不能修改。与该类相比，通常应该优先使用 java.lang.StringBuilder 类，因为它支持所有相同的操作，但由于它不执行同步，所以速度更快。为了获得更好的性能，在构造 StirngBuffer 或 StirngBuilder 时应尽可能指定它的容量。当然，如果操作的字符串长度不超过 16 个字符就不用了。相同情况下使用 StirngBuilder 比使用 StringBuffer 仅能获得 10%～15%左右的性能提升，但却要冒多线程不安全的风险。而在现实的模块化编程中，负责某一模块的程序员不一定能清晰地判断该模块是否会放入多线程的环境中运行，因此，除非能确定系统的瓶颈是在 StringBuffer 上，并且确定模块不会运行在多线程模式下，否则还是用 StringBuffer 为好。

13.5.2 程序性能优化

传统手机软件是由手机开发厂商固化在手机中的。在 SUN 公司推出 J2ME 后，各大手机厂商很快就推出了支持 J2ME 的手机。支持 J2ME 的手机可运行由第三方提供的基于 J2ME 开发的软件，手机的扩展功能得到极大增强。

在 J2ME 规范之中，J2ME 定义了基于 Java 类库的 CDC(Connected Device Configuration)和 CLDC(Connected LimitedDevice Configuration)，在 CDC 和 CLDC 之上，J2ME 又定义了 Profile 层，称为 MIDP(Mobile Information Device Profile)。MIDP 对 CDC / CLDC 进行了一定程度的封装，并定义了一整套应用程序接口和用户接口。MIDP 为 J2ME 应用提供了两类 UI(User Interface)，分别称作高级用户界面和低级用户界面。高级用户界面是被适配到设备上由手机操作系统定义外观的通用图形组件；低级用户界面则允许开发者根据需要在界面上任意绘制图形，是由开发者完全控制显示内容的图形界面。由于手机游戏界面绝大多数由自定义的图形元素构成，所以不可避免地要采用低级用户界面。

低级用户界面开发具有高度的自由性，不同的游戏架构和不同的编码风格将对最终产品的性能产生极大影响。目前对于 J2ME 手机游戏开发而言，最大的问题在于 J2ME 运行平台有限的资源。如何有效地利用现有资源以提高游戏的运行性能，成为开发者面临的首要问题。

目前被普遍采用的优化方案有。

1）优化循环，通过将重复的子表达式重新组织来提高循环体的运行性能。

2）减少使用对象的数量来提高运行性能。

3）缩减网络传输数据来缩短等待时间等。

在本节中将给出 3 种性能优化的策略。

1）采用对象池技术，提高对象的利用率。

2）局部使用基本数据类型代替对象，节省资源开销。

3）用简单的数值计算代替复杂的函数计算，节省处理器时间。

1. 采用对象池技术，提高对象的利用效率

Java 是面向对象的编程语言，创建和释放对象会占用相当大的资源，而在 Java 里不用对象在很多情况下又无法实现。本文提出一种对象池技术，将有效解决创建和释放对象带来的性能损失问题。

例如，游戏中敌机的处理方式：一种解决方案是游戏在载入关卡的时候，为每架敌机创建一个对象，随着游戏的行进，按照游戏进程显示不同的敌机。这种方案中创建对象的资源开销巨大，因此严重影响手机游戏的运行性能。虽然在敌机被击毁的时候可以将对应的对象设置为 null 并由 System.gc()回收，但在下一关卡载入的时候还要重新创建相关的对象，增加



了用户的等待时间。另一种方案是在游戏的进程中，根据需要动态创建敌机对象，被击毁后将对象设置为 null 并由 System.gc()回收。这种方案虽然能减少游戏载入时间，但是频繁地创建和释放对象的资源开销使游戏变得不流畅，对于射击等对实时性要求很高的游戏而言，这一点是不可接受的。

从研究数据来看，游戏性能损耗主要源之创建和释放对象，而不创建对象又无法实现逻辑功能，因此要尽量避免对象的创建和释放。问题的重点就转到怎样有效利用已有的对象上。本文提出的对象池技术，就是根据需求先创建一定量的对象，在需要创建对象的时候从池中申请空闲对象，释放对象时把对象释放回池中，以有效避免由创建和释放对象带来的性能损失。

分析游戏需求发现同时显示的敌机数量最多不过 5 架，采用对象池技术可以先定义一个对象池，容量为同时显示的敌机的最大数量。

```
Enemy[5]enemy=new Enemy[5];
for(int i=0; i<5; i++){
enemy[i]＝new Enemy();
}
```

在类 Enemy 里增加标志属性 used 和带参数的 reset 方法使对象可重置到初始状态，在载入游戏关卡的时候初始化对象池，在需要创建对象的时候从对象池获取一个未被使用的对象并使用 reset 方法初始化，需要释放对象的时候只需将标志位修改以供下次使用。与第一种解决方案相比使用对象池减少了相当大一部分的资源开销，与第二种解决方案相比使用对象池避免了频繁创建对象的额外资源开销。

2. 尽可能地使用基本数据类型代替对象

对象虽然屏蔽了细节实现，但是是以牺牲存储空间来实现的。使用基本数据类型则仅需要少量的存储空间。虽然对象在逻辑的实现上具有优势，但是在绝大多数情况下，使用基本数据类型比使用对象更高效，所以在局部不影响逻辑的情况下可以考虑用基本数据类型代替对象实现。

在游戏中飞机要发射子弹，基于面向对象的设计模式可将子弹抽象成 Bullet 类，然后定义代表子弹的对象池，在发射子弹的时候，在对象池中查找可用对象并重置其位置为飞机所在的位置，即在每一个 gameloop 中将 Bullet 对象的纵坐标值减少一定数值(屏幕坐标系 Y 轴坐标为自上而下递增)。由于对象操作的效率低于对基本数据类型的操作，由此可以想到由基本数据类型来代替对象实现，代码如下。

```
int[][] bullet=new int[2][10]; //假设同屏同时显示 10 颗子弹
for(int i=0; i<10; i++){
bullet[0][i]=999; //对应 Bullet 对象的 xposition 属性
bullet[1][i]=999; //对应 Bullet 对象的 yposition 属性
}
```

在每一个 gameloop 中代码如下。

```
while(run){
  for(int i=0; i<10; i++){
  if(bullet[1][i]==999){
```

```
//重置为发射子弹飞机当前的位置
bullet[0][i]==myPlane．getXposition()；
bullet[1][i]==myPlane．getYposition()；
}else{
//当前使用的子弹让其纵坐标减少
bullet[1][j]-=10：
}
}
}
```

3. 用简单的数值计算代替复杂的函数计算

在任何一款游戏里，都不可避免地要进行函数运算，而大量复杂的函数计算势必会占用大量的空间和处理器时间，进而影响游戏性能。减少函数计算复杂度或者减少复杂函数的调用次数甚至避免使用复杂函数计算，将有利于游戏性能的提高。与复杂函数运算相比，简单的数值计算无论从时间上还是在空间上都有极大的优势。

在游戏中敌机也会发射子弹，不同于我机子弹，敌弹的运行轨迹可以朝着任意一个方向。计算敌弹运行轨迹常用的方法是使用三角函数，根据飞行方向和横坐标轴正方向的夹角来计算敌弹的位移量。虽然 MIDP2.0 提供了三角函数的类库，但是在每个 gameloop 中计算每颗敌弹的运行轨迹势必有大量的三角函数运算而导致性能损失，当子弹数目变大时这种影响更加明显。同时由于 MIDP1.0 并不提供三角函数类库，所以该游戏无法运行在不支持 MIDP2.0 的手机上。

由基本的数学原理可知，当角度一定时其对应的三角函数值也是一定的，又因为三角函数是周期函数，因此本文的优化算法是用 0°～45°的函数值通过简单的四则运算来模拟任意函数值。下面介绍实现过程。

首先定义如下数组。

```
public static int iAngle[][]=new int[][]{
//根据游戏需要定义不同的精度，可以从 0°连续定义到 45°
//因为 SinX=Cos(90-X)，所以不必重复定义 46°～90°的函数值
   {0，1000，0}，
   {5，996，87}，
   {10，985，174)，
   {40，766，643}，
   }45，707，707}
   }
```

无论子弹朝着哪个方向飞行，假设飞行速度一定，其横纵坐标位移量均为该方向对应的余弦和正弦值，所以可以重新定义敌机子弹数组，将位移量包含到数组中。本游戏设定每颗子弹的飞行方向始终是固定不变的，即不会出现弧形的飞行轨迹。

```
int[][] enemyBullet=new int[4][20]；
// 假设同屏同时显示 20 颗敌机子弹
int size=enemyBultet．length；
for(int i=0：i
enemyBullet[0][i]=999；//对应敌机子弹对象的×坐标
```



```
enemyBullet[1][i]=999；//对应敌机子弹对象的 y 坐标
enemyBullet[2][i]=999；
// 对应敌机子弹对象的运行时的 x 坐标位移量
enemyBullet[3][i]=999；
// 对应敌机子弹对象的运行方的 y 坐标位移量
}
```

按照程序的需要，从数组 iAngle 中把特定的数值赋值到 enemyBullet[2][i]和 enemyBullet[3][i]，而在每一个 gameloop 中只需要执行如下代码。

```
inc size＝enemyBullet．length：
for(int i=0；i
 if(enemyBullet[0][i]!=999&&enemyBullet[1][i]!=999){
enemyBullet[0][i]+=enemyBullet[2][i]；
enemyBullet[1][i]+=enemyBullet[3][i]；
}
}
```

以上过程避免了每个 gameloop 中使用三角函数计算子弹的位移，同时也使 MIDP1.0 平台的手机不会因为不支持 MIDP2.0 而无法运行程序。为了保证足够的精度，这里相关函数值取 3 位小数，在处理时可以将横纵坐标值 enemyBullet[0][i]和 enemyBullet[1][i]定义成坐标值的 1000 倍，然后与坐标值 enemyBullet[2][i]和 enemyBullet[3][i]相加，显示的时候将 enemyBullet[0][i]和 enemyBullet[1][i]的值除以 1000 取整。

当前 J2ME 游戏的性能的瓶颈在于有限的存储资源和偏弱的处理器速度，而 J2ME 游戏优化的关键之处在于节省资源开销、节省冗余计算和计算简化，所以牺牲部分设计上的结构化(即面向对象)换来性能的提升，在当前情况下仍然是非常有必要的。本文提出的优化方法为在较低的硬件条件下实现流畅的画面，提供了可行的解决方案。

4．找出内存溢出元凶

在编写代码后有时候会抛 java.lang.OutofMemoryError 异常，就是 Java 的内存溢出。子窗体关闭后Java的垃圾回收机制无法回收其资源，因为这个Java程序可能要经常开关一些子窗体，那么这些子窗体关闭后无法释放资源就造成了 Java 程序 OutOfMemoryError 的潜在的隐患。子窗体关闭后资源无法释放的根本原因是：子窗体虽然调用了 dispose()方法，但是子窗体对象的引用或者是被静态 HashMap 引用，或者是它的内部子线程类没有释放，或者是它的某个事件监听类没有释放，需要彻底释放某个对象占用资源的关键在于找到并释放所有对它的引用。

程序中造成内存溢出可能性最大的是 HashMap，Hashtable 等集合类，尤其是静态的，更是要慎之又慎。它们引用的对象可能用户感觉已经销毁了，其实很可能忘记 remove 键值，而如果这些集合对象还是静态的挂在其他类里面，那么这个引用可能一直都在，借用 JProbe 测试一下，结果往往出人意料，解决办法：彻底删除 remove、clear 键，如果允许最好把集合对象设为 null。

对于不再使用的线程对象，如果要彻底杀了它，很多书上都推荐用 join 方法，这样做很可能要杀的线程仍旧好好的活在日益增大的内存里，很可能调用了线程的 sleep 方法后用 join 方法就会有点问题。解决办法：在 join 方法前再加一句执行 interrupt 方法，不过这个时候可能会有新的问题：执行 interrupt 方法后线程类会抛 InterruptedException，加一个开关变量做判断就能完美解决，可参考下面的代码。

```
    /**
     * <p>Description: 创建线程的内部类</p>
     * @author cuishen
     * @version 1.1
     */
    class NewThread implements Runnable {
        Thread t;
        NewThread() {
            t = new Thread(this, path);
            t.start();
        }
        public void run() {
            try {
                while(isThreadAlive) {
                    startMonitor();
                    Thread.sleep(Long.parseLong(controlList.get(controlList.size() - 1).toString()));
                }
            } catch (InterruptedException e) {
                if(!ifForceInterruptThread) {      //开关变量
                    stopThread(logThread);
                    String error = "InterruptedException！！！" + path + ": Interrupted，线程异常终止！
程序已试图重启该线程！！";
                    System.err.println(error);
                    LogService.writeLog(error);
                    createLogThread();
                }
            }
        }
    }

    public void createLogThread() {
        ifForceInterruptThread = false;         //开关变量
        logThread = new NewThread();
    }

    private void stopThread(NewThread thread) {
        try {
            thread.t.join(100);
        } catch (InterruptedException ex) {
            System.out.println("线程终止异常！！！");
        } finally {
            thread = null;
        }
    }

    /**
```



```
    * 关闭并彻底释放该线程资源的方法
    */
   public void stopThread() {
       try {
           ifForceInterruptThread = true;          //开关变量
           isThreadAlive = false;
           logThread.t.interrupt();
           logThread.t.join(50);
       } catch (InterruptedException ex) {
           System.out.println("线程终止异常！！！");
       } finally {
           this.controlList = null;
           this.keyList = null;
           logThread = null;
       }
   }
```

对于继承 JFrame 的窗体类，要注意在初始化方法中加入“this.setDefaultCloseOperation（DISPOSE_ON_CLOSE）;”，并且注意和其关联的事件监听类一律写成窗体类的内部类，这样窗体 dispose()的时候，这些内部类也一并销毁，就不会再有什么莫名其妙的引用了。

注意：JProbe 是一个监控 java 程序内存使用的工具。

13.5.3　何谓高效的Android

Android 设备是嵌入式设备。现代的手持设备，与其说是电话，更像一台拿在手中的电脑。但是，即使是“最快”的手持设备，其性能也赶不上一台普通的台式电脑。

这就是为什么在书写 Android 应用程序的时候要格外关注效率。这些设备并没有那么快，并且受电池电量的制约。这意味着，设备没有更多的能力，需要做的是必须把程序写得尽量有效。

对于占用资源的系统，有如下两条可遵循的基本原则。

- 不要做不必要的事。
- 不要分配不必要的内存。

所有的内容都要遵循这两个原则。

不可否认微优化（micro-optimization，代码优化，结构优化）的确会带来很多问题，诸如无法使用更有效的数据结构和算法。但是在手持设备上，别无选择。假如用户认为 Android 虚拟机的性能与台式机相当，程序很有可能一开始就占用了系统的全部内存（内存很小），这会让程序慢得像蜗牛一样，更别说做其他的操作了。

Android 的成功依赖于程序提供的用户体验。而这种用户体验，部分依赖于程序是响应快速而灵活的，还是响应缓慢而僵化的。因为所有的程序都运行在同一个设备之上，都在一起，这就如同在同一条路上行驶的汽车。而这篇文档就相当于在取得驾照之前必须要学习的交通规则。如果大家都按照这些规则去做，驾驶就会很顺畅，但是如果不这样做，可能会车毁人亡。这就是为什么这些原则十分重要。

不管 VM 是否支持实时（JIT）编译器（它允许实时地将 Java 解释型程序自动编译成本

机机器语言，以使程序执行的速度更快。有些 JVM 包含 JIT 编译器。)，下面提到的这些原则都是成立的。假如有目标完全相同的两个方法，在解释执行时 foo()比 bar()快，那么编译之后，foo()依然会比 bar()快。所以不要寄希望于编译器可以拯救程序。

（1）避免建立对象

世界上没有免费的对象。虽然 GC 为每个线程都建立了临时对象池，可以使创建对象的代价变得小一些，但是分配内存永远都比不分配内存的代价大。

如果在用户界面循环中分配对象内存，就会引发周期性的垃圾回收，用户就会觉得界面一顿一顿的。所以除非必要，应尽量避免建立对象的实例。下面的例子将帮助理解这条原则。

- 当从用户输入的数据中截取一段字符串时，尽量使用 substring 函数取得原始数据的一个子串，而不是为子串另外建立一份复件。这样就有一个新的 String 对象，它与原始数据共享一个 char 数组。
- 如果有一个函数返回一个 String 对象，而确切的知道这个字符串会被附加到一个 StringBuffer，那么，请改变这个函数的参数和实现方式，直接把结果附加到 StringBuffer 中，而不要再建立一个短命的临时对象。一个更极端的例子是，把多维数组分成多个一维数组。
- int 数组比 Integer 数组好，这也概括了一个基本事实，两个平行的 int 数组比(int,int)对象数组性能要好很多。同理，这适用于所有基本类型的组合。
- 如果想用一种容器存储(Foo,Bar)元组，尝试使用两个单独的 Foo[]数组和 Bar[]数组，一定比(Foo,Bar)数组效率更高。（也有例外的情况，就是当建立一个 API，让别人调用它的时候。这时候要注重对 API 借口的设计而牺牲一点速度。当然在 API 的内部，仍要尽可能地提高代码的效率）

总体来说，就是避免创建短命的临时对象。减少对象的创建就能减少垃圾收集，进而减少对用户体验的影响。

（2）使用本地方法

当在处理字串的时候，不要吝惜使用 String.indexOf(), String.lastIndexOf()等特殊实现的方法（specialty methods）。这些方法都是使用 C/C++实现的，比起 Java 循环快 10～100 倍。

（3）使用实类比接口好

假设有一个 HashMap 对象，可以将它声明为 HashMap 或者 Map。

```
Map myMap1 = new HashMap();
HashMap myMap2 = new HashMap();
```

究竟哪个更好呢？

按照传统的观点 Map 会更好些，因为这样可以改变他的具体实现类，只要这个类继承自 Map 接口。传统的观点对于传统的程序是正确的，但是它并不适合嵌入式系统。调用一个接口的引用会比调用实体类的引用多花费一倍的时间。

如果 HashMap 完全适合程序，那么使用 Map 就没有什么价值。如果有些地方不能确定，先避免使用 Map，剩下的交给 IDE 提供的重构功能好了。(当然公共 API 是一个例外，一个好的 API 常常会牺牲一些性能）

（4）用静态方法比虚方法好

如果不需要访问一个对象的成员变量，那么请把方法声明成 static。虚方法执行的更快，



因为它可以被直接调用而不需要一个虚函数表。另外也可以通过声明体现出这个函数的调用不会改变对象的状态。

（5）不用 getter 和 setter

在很多本地语言如 C++中，都会使用 getter（比如，i = getCount()）来避免直接访问成员变量（i = mCount）。在 C++中这是一个非常好的习惯，因为编译器能够内联访问，如果需要约束或调试变量，可以在任何时候添加代码。

在 Android 上，这就不是个好方法了。虚方法的开销比直接访问成员变量大得多。在通用的接口定义中，可以依照 OO 的方式定义 getters 和 setters，但是在一般的类中，应该直接访问变量。

（6）将成员变量缓存到本地

访问成员变量比访问本地变量慢得多，看下面一段代码。

```
for (int i = 0; i < this.mCount; i++) dumpItem(this.mItems[i]);
```

再好改成这样。

```
int count = this.mCount;
Item[] items = this.mItems;
for (int i = 0; i < count; i++) dumpItems(items[i]);          //使用 this 是为了表明这些是成员变量
```

另一个相似的原则是：永远不要在 for 的第二个条件中调用任何方法。如下面方法所示，在每次循环的时候都会调用 getCount()方法，这样做比在一个 int 先把结果保存起来开销大很多。代码如下。

```
for (int i = 0; i < this.getCount(); i++) dumpItems(this.getItem(i));
```

同样如果要多次访问一个变量，也最好先为它建立一个本地变量，代码如下。

```
protected void drawHorizontalScrollBar(Canvas canvas, int width, int height) {
if (isHorizontalScrollBarEnabled())
    {
        int size = mScrollBar.getSize(false);
        if (size <= 0) { size = mScrollBarSize;
        }
        mScrollBar.setBounds(0, height – size, width, height);
        mScrollBar.setParams( computeHorizontalScrollRange(), computeHorizontalScrollOffset(),
computeHorizontalScrollExtent(), false);
        mScrollBar.draw(canvas);
        }
        }
```

此处有 4 次访问成员变量 mScrollBar，如果将它缓存到本地，4 次成员变量访问就会变成 4 次效率更高的栈变量访问。

另外就是方法的参数与本地变量的效率相同。

（7）使用常量

看看这两段在类前面的声明。

```
static int intVal = 42;
static String strVal = “Hello, world!”;
```

其会生成一个叫做<clinit>的初始化类的方法，当类第一次被使用的时候这个方法会被执行。方法会将 42 赋给 intVal，然后把一个指向类中常量表的引用赋给 strVal。当以后要用到这些值的时候，会在成员变量表中查找到他们。下面做些改进，使用“final”关键字。

```
static final int intVal = 42;
static final String strVal = “Hello, world!”;
```

现在，类不再需要<clinit>方法，因为在成员变量初始化的时候，会将常量直接保存到类文件中。用到 intVal 的代码被直接替换成 42，而使用 strVal 的会指向一个字符串常量，而不是使用成员变量。

将一个方法或类声明为“final”不会带来性能的提升，但是会帮助编译器优化代码。举例说，如果编译器知道一个“getter”方法不会被重载，那么编译器会对其采用内联调用。

也可以将本地变量声明为“final”，同样，这也不会带来性能的提升。使用“final”只能使本地变量看起来更清晰些（但是有些时候这是必须的，比如，在使用匿名内部类的时候）

（8）谨慎使用 foreach

foreach 可以用在实现了 Iterable 接口的集合类型上。foreach 会给这些对象分配一个 iterator，然后调用 hasNext()和 next()方法。最好使用 foreach 处理 ArrayList 对象，但是对其他集合对象，foreach 相当于使用 iterator。

下面展示了 foreach 一种可接受的用法。

```
public class Foo { int mSplat; static Foo mArray[] = new Foo[27];
  public static void zero() {
  int sum = 0;
  for (int i = 0; i < mArray.length; i++) { sum += mArray[i].mSplat;
  }
  }
  public static void one() {
  int sum = 0;
  Foo[] localArray = mArray;
  int len = localArray.length;
  for (int i = 0;
  i < len; i++) { sum += localArray[i].mSplat;
  }
  }
  public static void two() {
  int sum = 0;
  for (Foo a: mArray) { sum += a.mSplat;
  }
  }
  }
```

分析上述代码。



- 在 zero()中，每次循环都会访问两次静态成员变量，取得一次数组的长度。
- 在 one()中，将所有成员变量存储到本地变量。
- 在 two()中，使用了在 java1.5 中引入的 foreach 语法。编译器会将对数组的引用和数组的长度保存到本地变量中，这对访问数组元素非常好。但是编译器还会在每次循环中产生一个额外的对本地变量的存储操作（对变量 a 的存取），这样会比 one()多出 4 个字节，速度要稍微慢一些。

综上所述：foreach 语法在运用于 array 时性能很好，但是运用于其他集合对象时要小心，因为它会产生额外的对象。

（9）避免使用枚举

枚举变量非常方便，但不幸的是它会牺牲执行的速度并大幅增加文件体积。例如：

```
public class Foo {
  public enum Shrubbery {
  GROUND, CRAWLING, HANGING
  }
  }
```

会产生一个 900 字节的.class 文件(Foo$Shubbery.class)。在它被首次调用时，这个类会调用初始化方法来准备每个枚举变量。每个枚举项都会被声明成一个静态变量，并被赋值。然后将这些静态变量放在一个名为“$VALUES”的静态数组变量中。而这么一大堆代码，仅仅是为了使用三个整数。如果：

```
Shrubbery shrub = Shrubbery.GROUND;
```

会引起一个对静态变量的引用，如果这个静态变量是 final int，那么编译器会直接内联这个常数。

使用枚举变量可以让 API 更出色，并能提供编译时的检查。所以在通常的时候毫无疑问应该为公共 API 选择枚举变量。但是当性能方面有所限制的时候，就应该避免这种做法。

在有些情况下，使用 ordinal()方法获取枚举变量的整数值会更好一些，举例来说，将：

```
for (int n = 0; n < list.size(); n++) { if (list.items[n].e == MyEnum.VAL_X) // do stuff 1 else if
(list.items[n].e == MyEnum.VAL_Y) // do stuff 2 }
```

替换为：

```
int valX = MyEnum.VAL_X.ordinal();
int valY = MyEnum.VAL_Y.ordinal(); int count = list.size();
MyItem items = list.items();
for (int n = 0; n < count; n++) { int valItem = items[n].e.ordinal();
if (valItem == valX) // do stuff 1
else if (valItem == valY) // do stuff 2
}
```

这样会使性能得到一些改善，但这并不是最终的解决之道。将与内部类一同使用的变量声明在包范围内。看下面的类定义。

```
public class Foo {
  private int mValue;
  public void run() {
  Inner in = new Inner();
  mValue = 27; in.stuff();
  }
  private void doStuff(int value) {
  System.out.println(”Value is ” + value); }
  private class Inner {
  void stuff() {
  Foo.this.doStuff(Foo.this.mValue);
  }
  }
  }
```

这其中的关键是，定义了一个内部类(Foo$Inner)，它需要访问外部类的私有域变量和函数。这是合法的，并且会打印出希望的结果“Value is 27”。

问题是在技术上来讲（在幕后），Foo$Inner 是一个完全独立的类，它要直接访问 Foo 的私有成员是非法的。要跨越这个鸿沟，编译器需要生成一组方法。

```
static int Foo.access$100(Foo foo) {
return foo.mValue;
}
static void Foo.access$200(Foo foo, int value) {
foo.doStuff(value);
}
```

内部类在每次访问“mValue”和“doStuff”方法时，都会调用这些静态方法。就是说，上面的代码说明了一个问题，是在通过接口方法访问这些成员变量和函数而不是直接调用它们。在前面我们已经说过，使用接口方法（getter、setter）比直接访问速度要慢。所以这个例子就是在特定语法下面产生的一个“隐性的”性能障碍。

通过将内部类访问的变量和函数声明由私有范围改为包范围，可以避免这个问题。这样做可以让代码运行更快，并且避免产生额外的静态方法。（遗憾的是，这些域和方法可以被同一个包内的其他类直接访问，这与经典的 OO 原则相违背。因此当设计公共 API 的时候应该谨慎使用这条优化原则）

（10）避免使用浮点数

在奔腾 CPU 出现之前，游戏设计者做得最多的就是整数运算。随着奔腾的到来，浮点运算处理器成为了 CPU 内置的特性，浮点和整数配合使用，能够让游戏运行得更顺畅。通常在桌面电脑上，可以随意地使用浮点运算。

但是非常遗憾，嵌入式处理器通常没有支持浮点运算的硬件，所有对”float”和”double”的运算都是通过软件实现的。一些基本的浮点运算，甚至需要毫秒级的时间才能完成。

甚至是整数，一些芯片有对乘法的硬件支持而缺少对除法的支持。这种情况下，整数的除法和取模运算也是有软件来完成的。所以当在使用哈希表或者做大量数学运算时一定要小



心谨慎。

13.5.4　Android上的单元测试

任何程序的开发都离不开单元测试来保证其健壮和稳定，Android 的程序自然也不例外。从 Android SDK 0.9 开始，就有了比较成熟的测试框架。接下来，对此内容做一下梳理和总结。

（1）JUnit 还能用么

在 Java 下做单元测试必然用到 JUnit。这里说的 JUnit 是指从 Apache 基金会下载的 junit.jar 里提供的一系列单元测试功能。这些功能显然是运行在 JDK 之上的。在 Android 下已经没有了 JDK，自然也无法运行 JUnit。但是这并不妨碍用户利用 JUnit 编写单元测试。只不过在运行单元测试时，一定要用 JDK 来运行，利用 Java 命令来启动 JUnit 的某个 Runner。如果是用 Eclipse 的话，可以在 Run Configuration 里新建一个 JUnit。但是一定要记得在 Classpath 选项卡里将 Bootstrap Entries 中的 Android Library 改成 JRE，并且添加 junit.jar，如图 13-5 所示。

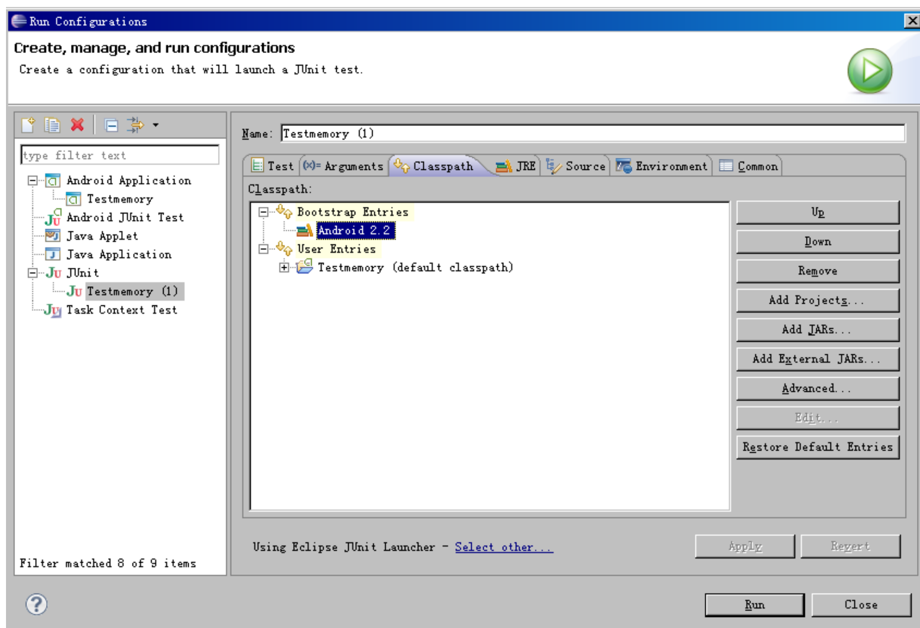


图 13-5　JUnit

很明显的，这种测试就是正规的 Java 单元测试，和 Android 没有任何关系。用户无法测试任何关于 Android 系统中的 API，如 Activity、人机界面等。所以，如果想测试仅仅是一些封装数据的对象，或者是纯粹的数值计算，还是可以用这种方法的。

（2）junit.framework

当大家看到这个包的时候，第一反应是 Android 是不是已经完整集成了 JUnit。很遗憾这不是事实。如果按照 JUnit 的运行方法，却不像上面那样改用 JDK，就一定会得到如下异常。

```
#
# An unexpected error has been detected by Java Runtime Environment:
#
```

```
# Internal Error (classFileParser.cpp:2924), pid=4900, tid=4476
#Error: ShouldNotReachHere()
#
# Java VM: Java HotSpot(TM) Client VM (10.0-b19 mixed mode windows-x86)
# An error report file with more information is saved as:
# E:\Mydoc\EclipseWorkspace\TestAndroid\hs_err_pid4900.log
#
# If you would like to submit a bug report, please visit:
# http://java.sun.com/webapps/bugreport/crash.jsp
#
```

实际上，TestCase 类用于在 Android 中担当所有独特的 TestCase 的基类，它是一个 Abstract Class。里面有很多以“TestCase”为后缀的类。之所以有那么多 TestCase 主要是为了简化工作。例如，当想对一个访问数据库的功能进行测试时，首先需要自己启动并初始化数据库。在这里是类似的，如果想测试一个 Activity，首先要启动它。而 ActivityTestCase 就会自动做完这些事情。而 ActivityUnitTestCase 会更注重测试的独立性，它会让测试与 Android 底层的联系降到最低。其余的类可以查看相关的 Javadoc 来按需挑选。要编写测试，就是找到合适的 XXXTestCase 作为基类来继承，并且编写自己的测试方法。

很明显的，最简单的编写测试的方法就是继承 AndroidTestCase 写一个自己的 TestCase。然后为自己的一组 TestCase 写一个 Activity 界面，由界面控制 TestCase 的启动，运行和结果报告。但是，用户很快会发现，为何要给测试写一个界面呢？这太诡异了。这时就需要一种技术，它可以利用命令行（Shell）来启动一组测试，并且通过命令行的形式给出结果。这就是所谓的 Instrumentation。

（3）Instrumentation

一般在开发 Android 程序的时候，需要写一个 manifest 文件，其结构如下。

```
<application android:icon=”@drawable/icon” android:label=”@string/app_name”>
<activity android:name=”.TestApp” android:label=”@string/app_name”>
……
</activity>
</application>
```

这样，在启动程序的时候就会先启动一个 Application，然后在此 Application 运行过程中根据情况加载相应的 Activity，而 Activity 是需要一个界面的。但是 Instrumentation 并不是这样的。用户可以将 Instrumentation 理解为一种没有图形界面的、具有启动能力的、用于监控其他类（用 Target Package 声明）的工具类。任何想成为 Instrumentation 的类必须继承 android.app.Instrumentation。

对于单元测试，需要认真了解的就是 android.test.InstrumentationTestRunner 类。这是 Android 单元测试的主入口。它相当于 JUnit 当中 TestRunner 的作用。

那么如何加载它呢，首先要在 manifest 文件中加入一行关于 Instrumentation 的声明。比如 Android Api Demos 中的测试里的 manifest 如下。

```
<manifest xmlns:android=”http://schemas.android.com/apk/res/android”
```



```
package="com.example.android.apis.tests">
<application>
<uses-library android:name="android.test.runner" />
</application>
<instrumentation android:name="android.test.InstrumentationTestRunner"
android:targetPackage="com.example.android.apis"
android:label="Tests for Api Demos."/>
</manifest>
```

当编辑好 manifest，就可以打包（build，可以用 Eclipse ADT 来做，也可以用 aapt 命令手工完成），然后安装到虚拟机上（用 adb install 命令）。之后就可以利用命令行的方式来加载单元测试了。在 Android Shell 中加载一个 Instrumentation 的方法是利用以下命令。

```
adb shell am instrument –w XXXXXX
```

其中-w 是指定 Instrumentation 类的参数标志。下面是一个简单的例子。

```
adb shell am instrument -w com.android.foo/android.test.InstrumentationTestRunner
```

当然，也可以利用 adb shell 先进入 Android 命令行模式，再直接写 am instrument －w XXXXXXX。

（4）如何在 Android 中利用 Instrumentation 来进行测试

在介绍具体的命令之前，先了解一下单元测试的层次。一组单元测试可以被组织成若干个 TestSuite。每个 TestSuite 包含若干 TestCase（某个继承 android.jar 的 junit.framework.TestCase 的类）。每个 TestCase 又包含若干个 Test（具体的 test 方法）。

如果假设 com.android.foo 是测试代码的包的根。当执行以下命令时，会执行所有的 TestCase 的所有 Test。测试的对象就是在 Target Package 中指定的包中的代码。

```
adb shell am instrument -w com.android.foo/android.test.InstrumentationTestRunner
```

如果想运行一个 TestSuite，首先继承 android.jar 的 junit.framework.TestSuite 类，实现一个 TestSuite（比如叫 com.android.foo.MyTestSuite），然后执行以下命令执行此 TestSuite。

```
adb shell am instrument -e class com.android.foo.MyTestSuite -w com.android.foo/android. test.
InstrumentationTestRunner
```

其中的-e 表示额外的参数，语法格式如下。

```
-e [arg1] [value1] [arg2] [value2]
```

在此用到了 class 参数。

如果仅仅想运行一个 TestCase（比如叫 com.android.foo.MyTestCase），则用以下命令。

```
adb shell am instrument -e class com.android.foo.MyTestCase -w com.android.foo/android.test.
InstrumentationTestRunner
```

如果仅仅想运行一个 Test（比如就是上面 MyTestCase 的 testFoo 方法），很类似的，就用以下命令。

```
adb shell am instrument -e class com.android.foo.MyTestCase#testFoo -w com.android.foo/android.test.
InstrumentationTestRunner
```

然后，所有的测试结果会输出到控制台，并会做一系列统计，如标记为 E 的是 Error，标记为 F 的是 Failure，Success 的测试则会标记为一个点。这和 JUnit 的语义一致。如果希望断点调试测试，只需要直接在代码上加上断点，然后将运行命令参数的-e 后边附加上 debug true 后运行即可。更加详细的内容可以看 InstrumentationTestRunner 的 Javadoc。

（5）如何在 Android 的单元测试中做标记

在 android.test.annotation 包里定义了几个 annotation，包括@LargeTest、@MediumTest、@SmallTest、@Smoke 和@Suppress。用户可以根据自己的需要用这些 annotation 来对自己的测试分类。在执行单元测试命令时，可以在-e 参数后设置“size large”/“size medium”/“size small”来执行具有相应标记的测试。特别的@Supperss 可以取消被标记的 Test 的执行。

13.6　UI界面优化

很多程序员急功近利，为了实现某些功能，只要编写出对应的代码即可，也不考虑具体的性能。但这样与追求精益求精的思想是背道而驰的，往往就是因为满足于一个结果，而放弃探求更加优化的处理方法。

当关注应用程序或者游戏所达到的结果时，往往非常容易忽视一些优化的问题，例如，内存优化，线程优化，Media 优化和 UI 优化等。不同的模块都存在更为巧妙的方式来对待一般性问题，所以每当实现一个行为后，稍微多花一些时间来考虑目前所做的工作是否存在更为高效的解决办法。

在 Android 中，LinearLayout 表示 UI 的框架，而且也是最直观和方便的方法。例如，创建一个 UI 用于展现 Item 的基本内容，如图 13-6 所示。将图形用一个线框形式表示，如图 13-7 所示。

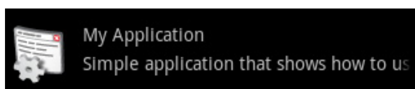


图 13-6　LinearLayout 布局



图 13-7　线框图

接可以通过 LinearLayout 来快速实现这个 UI 的排列，代码如下。

```
<LinearLayout xmlns:
    android="http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="?android:attr/listPreferredItemHeight"
    android:padding="6dip">
    <ImageView
        android:id="@+id/icon"
        android:layout_width="wrap_content"
        android:layout_height="fill_parent"
        android:layout_marginRight="6dip"
        android:src="@drawable/icon" />
```



```
        <LinearLayout
            android:orientation="vertical"
            android:layout_width="0dip"
            android:layout_weight="1"
            android:layout_height="fill_parent">
            <TextView
                android:layout_width="fill_parent"
                android:layout_height="0dip"
                android:layout_weight="1"
                android:gravity="center_vertical"
                android:text="My Application" />
            <TextView
                android:layout_width="fill_parent"
                android:layout_height="0dip"
                android:layout_weight="1"
                android:singleLine="true"
                android:ellipsize="marquee"
                android:text="Simple application that shows how to use RelativeLayout" />
        </LinearLayout>
    </LinearLayout>
```

虽然可以通过 Linearlayout 实现用户所预想的结果，但是在这里存在一个优化的问题，尤其是针对大量 Items。比较 RelativeLayout 和 LinearLayout，在资源利用上前者会占用更少的资源而达到相同的效果，下面是用 RelativeLayout 实现的代码。

```
    <RelativeLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:layout_width="fill_parent"
        android:layout_height="?android:attr/listPreferredItemHeight"
        android:padding="6dip">

        <ImageView
            android:id="@+id/icon"
            android:layout_width="wrap_content"
            android:layout_height="fill_parent"
            android:layout_alignParentTop="true"
            android:layout_alignParentBottom="true"
            android:layout_marginRight="6dip"

            android:src="@drawable/icon" />

        <TextView
            android:id="@+id/secondLine"
            android:layout_width="fill_parent"
            android:layout_height="26dip"
            android:layout_toRightOf="@id/icon"
            android:layout_alignParentBottom="true"
            android:layout_alignParentRight="true"
```

```
                android:singleLine="true"
                android:ellipsize="marquee"
                android:text="Simple application that shows how to use RelativeLayout" />

        <TextView
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:layout_toRightOf="@id/icon"
                android:layout_alignParentRight="true"
                android:layout_alignParentTop="true"
                android:layout_above="@id/secondLine"
                android:layout_alignWithParentIfMissing="true"
                android:gravity="center_vertical"
                android:text="My Application" />
    </RelativeLayout>
```

针对RelativeLayout有一点需要注意，因为它内部是通过多个View之间的关系而确定的框架，那么当其中某一个View因为某些需要调用GONE来完全隐藏掉后，会影响与其相关联的Views。Android提供了一个属性 alignWithParentIfMissing，用于解决类似问题，当某一个View无法找到与其相关联的Views后将依据alignWithParentIfMissing的设定判断是否与父级View对齐。

图 13-8 和图 13-9 是两种不同 layout 在 Hierarchy Viewer 中的层级关系图。

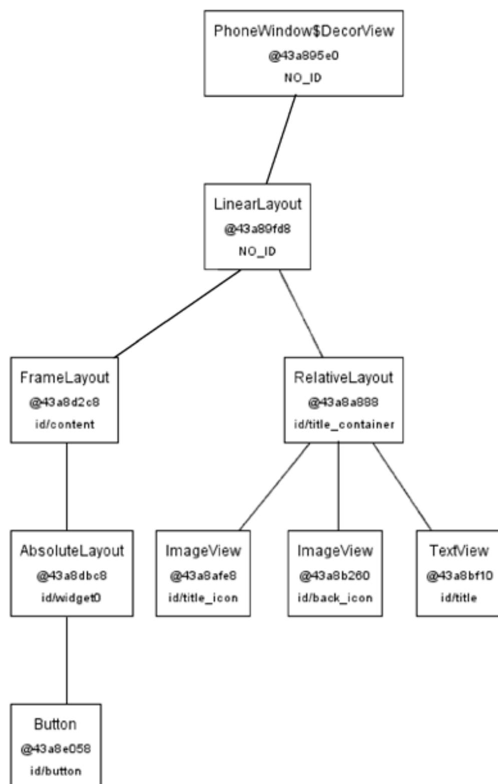


图 13-8　layout 在 Hierarchy Viewer 中的层级关系图



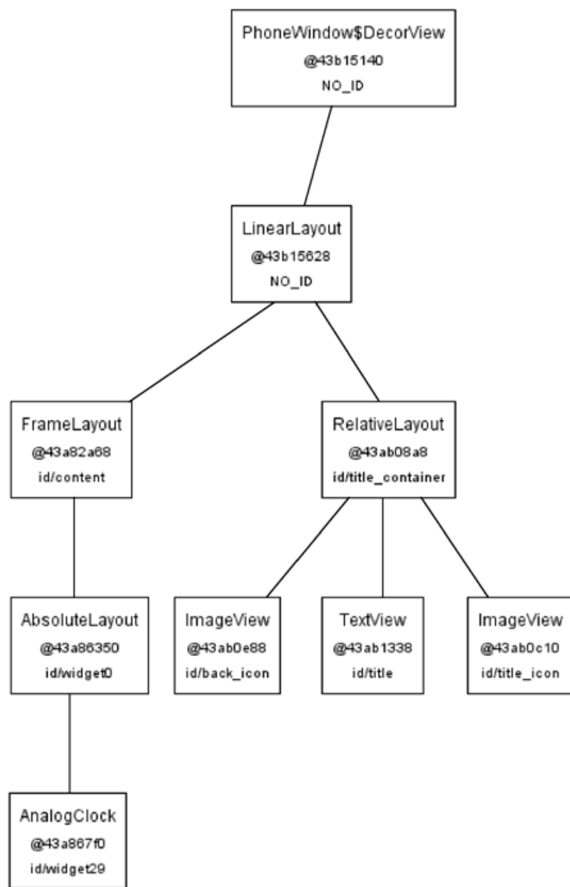


图 13-9　layout 在 Hierarchy Viewer 中的层级关系图

简单或复杂的问题都需要时常考虑如何优化资源的分配。比如一个功能很简单的应用程序，它会调用一些常用的对话框或者输入面板，这需要采用统一的方式来针对不同的应用程序制定统一标准。

当开始 Android UI 优化时，有必要继续考虑资源复用。手机开发给用户的直观感觉是运行其上的软件应该尽可能地达到资源高效利用的极致，而不能像开发 PC 机那样，似乎有用之不尽的资源。

定义 Android Layout(XML)时，有 4 个比较特别的标签是非常重要的，其中有 3 个是与资源复用有关，分别是<viewStub/>，<requestFocus/>，<merge/>和<include/>。可是以往所接触的案例或者官方文档的例子都没有着重去介绍这些标签的重要性。

1）<viewStub />：此标签可以使 UI 在特殊情况下，直观效果类似于设置 View 的不可见性，但是其更大的意义在于被这个标签所包裹的 Views 在默认状态下不会占用任何内存空间。viewStub 通过 include 从外部导入 Views 元素。

其用法是通过 android:layout 来指定所包含的内容。默认情况下，ViewStub 所包含的标签都属于 visibility=GONE。viewStub 通过方法 inflate()来召唤系统加载其内部的 Views。例如：

```
<ViewStub android:id="@+id/stub"
```

```
android:inflatedId="@+id/subTree"
android:layout="@layout/mySubTree"
android:layout_width="120dip"
android:layout_height="40dip" />
```

2）<merge />：通过删减多余或额外的层级，达到优化整个 Android Layout 结构的目的。

3）<include />：可以通过这个标签直接加载外部的 xml 到当前结构中，是复用 UI 资源的常用标签。其用法是将需要复用 xml 文件路径赋予 include 标签的 Layout 属性。例如：

```
<include android:id="@+id/cell1" layout="@layout/ar01" />
<include android:layout_width="fill_parent" layout="@layout/ar02" />
```

4）<requestFocus />：标签用于指定屏幕内的焦点 View，其用法是将标签置于 Views 标签内部。例如：

```
<EditText id="@+id/text"
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:layout_weight="0"
        android:paddingBottom="4">
    <requestFocus />
</EditText>
```

单独将<merge />标签做个介绍，是因为它在优化 UI 结构时起到很重要的作用。目的是通过删减多余或者额外的层级，从而优化整个 Android Layout 的结构。

下面通过一个例子来了解这个标签实际所产生的作用，这样可以更直观地了解<merge/>的用法。本实例代码保存在“光盘:\daima\13\”中，命名为 Examples。下面开始介绍本实例的具体实现过程。

第 1 步：建立一个简单的 Layout，其中包含两个 Views 元素：ImageView 和 TextView 默认状态下将这两个元素放在 FrameLayout 中。其效果是在主视图中全屏显示一张图片，之后将标题显示在图片上，并位于视图的下方。实现文件 main.xml 的具体实现代码如下所示：

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout
      xmlns:android="http://schemas.android.com/apk/res/android"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      >
      <ImageView
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:scaleType="center"
            android:src="@drawable/golden_gate"
            />
```



```
            <TextView
                android:layout_width="wrap_content"
                android:layout_height="wrap_content"
                android:layout_marginBottom="20dip"
                android:layout_gravity="center_horizontal|bottom"
                android:padding="12dip"
                android:background="#AA000000"
                android:textColor="#ffffffff"
                android:text="Golden Gate"
                />
    </FrameLayout>
```

此时执行后的效果如图 13-10 所示。

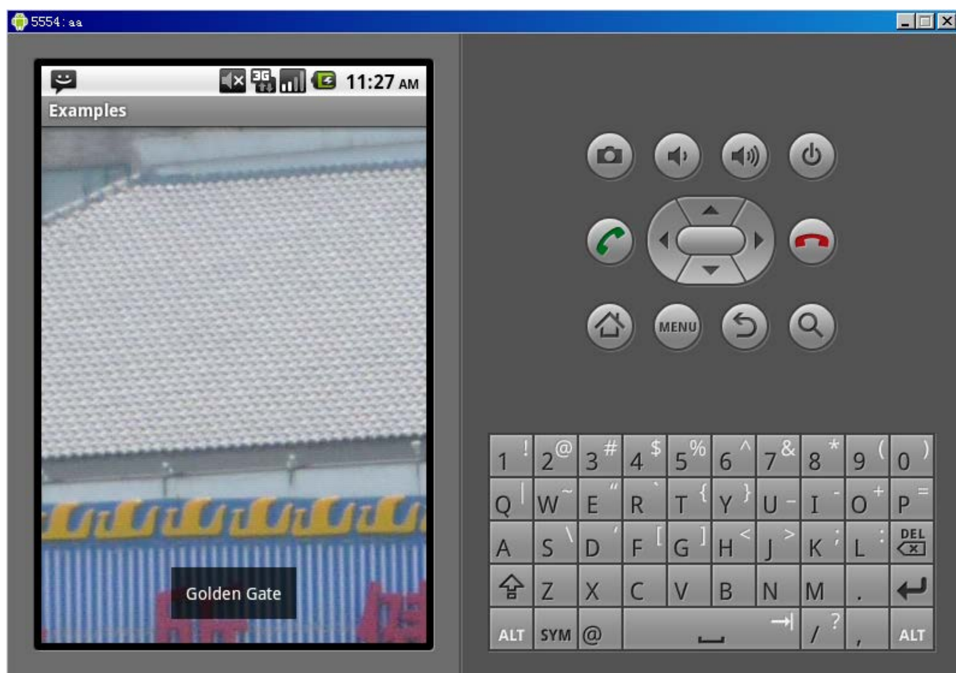


图 13-10　执行效果

第 2 步：启动 SDK 目录下的“tools”文件夹中的 hierarchyviewer.bat，如图 13-11 所示。

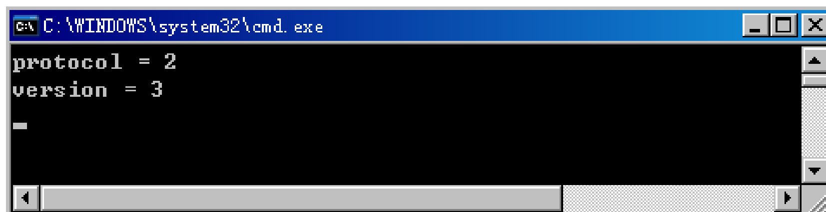


图 13-11　启动 hierarchyviewer.bat

此时可以查看当前 UI 的结构视图，如图 13-12 所示。



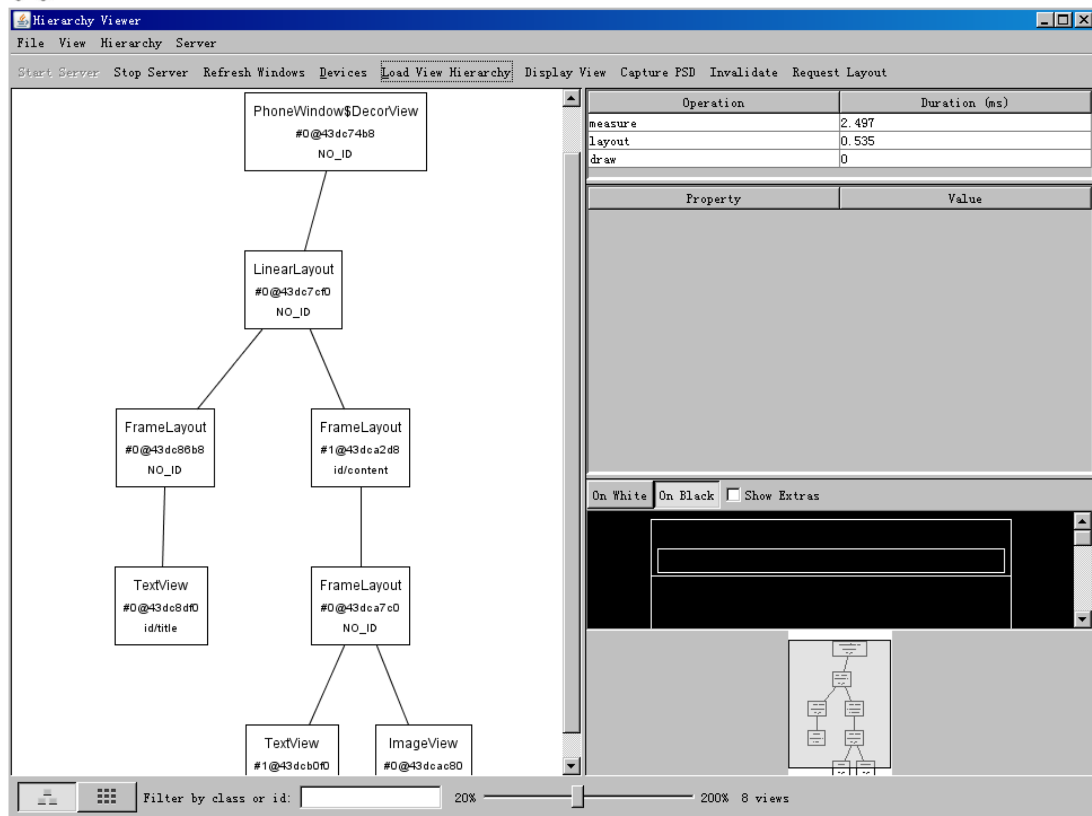


图 13-12 文件 main.xml 的 UI 结构视图

此时可以很明显地看到由红色线框所包含的结构出现了两个 framelayout 节点，很明显这两个完全意义相同的节点造成了资源浪费（这里可以提醒大家在开发工程中可以习惯性的通过 hierarchyViewer 查看当前 UI 资源的分配情况），那么如何才能解决这种问题呢（就当前例子是如何去掉多余的 frameLayout 节点）？这时候就要用到<merge />标签来处理类似的问题了。

第 3 步：将上边 xml 代码中的 framLayout 替换成 merge，实现文件 main2.xml 的具体实现代码如下所示。

```
<merge
    xmlns:android="http://schemas.android.com/apk/res/android"
    >
    <ImageView
        android:layout_width="fill_parent"
        android:layout_height="fill_parent"
        android:scaleType="center"
        android:src="@drawable/golden_gate"
        />
    <TextView
        android:layout_width="wrap_content"
```



```
                android:layout_height="wrap_content"
                android:layout_marginBottom="20dip"
                android:layout_gravity="center_horizontal|bottom"
                android:padding="12dip"
                android:background="#AA000000"
                android:textColor="#ffffffff"
                android:text="Golden Gate"
                />
</merge>
```

此时运行程序后，在 Emulator 中显示的效果是一样的，可是通过 hierarchyviewer 查看的 UI 结构是有变化的，如图 13-13 所示。

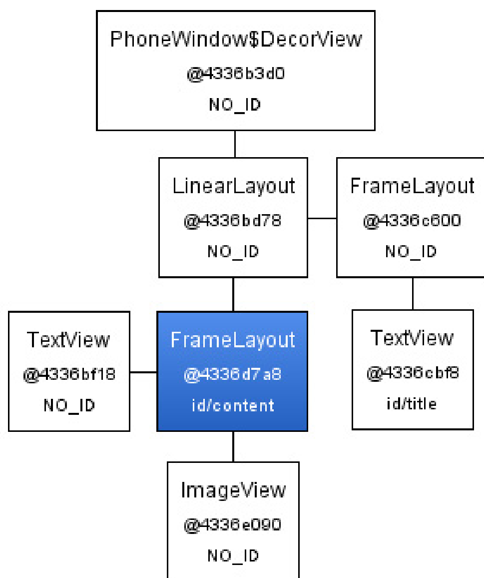


图 13-13　UI 结构视图

当初多余的 FrameLayout 节点被合并在一起了，或者可以理解为将 merge 标签中的子集直接加到 Activity 的 FrameLayout 跟节点下（这里需要提醒大家注意：所有的 Activity 视图的根节点都是 frameLayout）。如果所创建的 Layout 并不是用 framLayout 作为根节点（而是应用 LinerLayout 等定义 root 标签），就不能应用上边的例子通过 merge 来优化 UI 结构。

除了上边的例子外，meger 还有另外一个用法。当应用 Include 或者 ViewStub 标签从外部导入 xml 结构时，可以将被导入的 xml 用 merge 作为根节点表示，这样当被嵌入父级结构中后可以很好地将它所包含的子集融合到父级结构中，而不会出现冗余的节点。

读者有两点需要特别注意。

- <merge />只可以作为 xml layout 的根节点。
- 当需要扩充的 xml layout 本身是由 merge 作为根节点的话，需要将被导入的 xml layout 置于 viewGroup 中，同时需要设置 attachToRoot 为 True。

第 14 章　网络RSS阅读器

RSS 也叫聚合 RSS 是在线共享内容的一种简易方式（也叫聚合内容，Really Simple Syndication）。通常在时效性比较强的内容上使用 RSS 订阅能更快速获取信息，网站提供 RSS 输出，有利于让用户获取网站内容的最新更新。本章将讲解在 Android 手机中获取指定 RSS 在线信息的方法。

14.1　基础知识介绍

在学习本实例之前，需要预先了解与其相关的几个基本知识。将详细介绍 RSS 的主要用途、RSS 阅读器和 RSS 语法，帮助读者的知识更上一层楼。

14.1.1　RSS的用途

RSS 的主要用途如下。

1）可以订阅 BLOG（可以订阅工作中所需的技术文章；也可以订阅与自己有共同爱好的作者的 Blog，总之，对什么感兴趣就可以订什么）。

2）订阅新闻（无论是奇闻怪事、明星消息、体坛风云，只要想知道的，都可以订阅）。

3）再也不用一个网站一个网站，一个网页一个网页去逛了。只要将需要的内容订阅在一个 RSS 阅读器中，这些内容就会自动出现在阅读器里，也不必为了一个急切想知道的消息而不断地刷新网页，因为一旦有了更新，RSS 阅读器就会自己通知用户。

其实订阅 RSS 新闻内容要先安装一个 RSS 阅读器。然后将提供 RSS 服务的网站加入到 RSS 阅读器的频道即可。具体如下。

1）选择有价值的 RSS 信息源。

2）启动 RSS 订阅程序，将信息源添加到自己的 RSS 阅读器或者在线 RSS。

3）接收并获取定制的 RSS 信息。

14.1.2　RSS阅读器

目前，RSS 阅读器基本可以分为 3 类。

第一类大多数阅读器是运行在计算机桌面上的应用程序，通过所订阅网站的新闻供应，可自动、定时地更新新闻标题。在该类阅读器中，有 Awasu、FeedDemon 和 RSSReader 这 3 款流行的阅读器，都提供免费试用版和付费高级版。国内最近也推出了几款 RSS 阅读器：周博通、看天下、博阅。另外，开源社区也推出了很多优秀的阅读器，RSSOWl（完全 Java 开发）它不仅是完全支持中文界面，而且还是完全的免费软件。（后面就将以开源软件周博通和 RSSOWl 为例，为大家介绍怎样来使用 RSS 阅读器—周伯通，怎样使用 RSS 阅读器—RSSOWl）



第二类新闻阅读器通常是内嵌于已在计算机中运行的应用程序中。例如，NewsGator 内嵌在微软的 Outlook 中，所订阅的新闻标题位于 Outlook 的收件箱文件夹中。另外，Pluck 内嵌在 Internet Explorer 浏览器中。

第三类则是在线的 WEB RSS 阅读器，其优势在于不需要安装任何软件就可以获得 RSS 阅读的便利，并且可以保存阅读状态，推荐和收藏自己感兴趣的文章。提供此服务的有两类网站，一种是专门提供 RSS 阅读器的网站，如鲜果、抓虾；另一种是提供个性化首页的网站，如国外的 netvibes、pageflakes，国内的雅蛙、阔地。

14.1.3　RSS语法

RSS 2.0 的语法规则非常简单并十分的严格，看下面的代码。

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<rss version="2.0">
<channel>

<title>W3Schools</title>
<link>http://www.w3schools.com</link>
<description>W3Schools Web Tutorials </description>

<item>
<title>RSS Tutorial</title>
<link>http://www.w3schools.com/rss</link>
<description>Check out the RSS tutorial
on W3Schools.com</description>
</item>

</channel>
</rss>
```

其中，<channel>元素内是描述 RSS feed 的地方。

RSS 的<channel>元素是项目内容显示的地方。它就像 RSS 的标题。一般来讲它不会频繁的改动。有 3 个内部元素是必须有的，分别是<title>、<link>和<description>。具体说明如下。

- <title>元素里应该包含用户的站和 RSS feed 简短的说明。
- <link>元素应该定义网站主页的链接。
- <description>元素应该描述用户的 RSS feed。

<channel>内的可选元素如下。

- <category>定义一个或多个频道分类。
- <cloud>允许更新通告。
- <copyright>提醒有关版权。
- <docs>频道所使用的 RSS 版本文档 URL。
- <generator>如果频道是自动生成器产生的，就在这里定义。
- <image>给频道加图片。

- <language>描述了频道所使用的语言。
- <lastBuildDate>定义频道最新一次改动的时间。
- <managingEditor>定义编辑站点人员的 E-mail 地址。
- <pubDate>定义频带最新的发布时间。
- <rating>页面评估。
- <ttl>存活的有效时间。
- <webMaster>定义站长的邮件地址。

<item>元素内是网站连接和描述更新内容的地方。<item>是显示 RSS 更新内容的地方，它像是文章的标题。当站点有更新时，RSSfeed 中的<item>元素就会被建立起来。<item>元素里有几个可选的元素，但<title>和<description>是必须有的。

一个 RSS 的<item>应该包括：<title>、<link>和<description>元素。

- <title>元素是项目的题目，应该用十分简短的描述。
- <link>元素项目所关联的连接。
- <description>元素就是 RSS feed 的描述部分，这应该是描述用户的 RSS feed 项目的。

可选的<item>元素如下。

- <author>定义作者。
- <category>类别。
- <comments>针对项目的评论页 URL。
- <enclosure>描述一个与项目有关的媒体对象。
- <guid>针对项目定义独特的标志。
- <pubDate>项目发布时间。
- <source>转载地址（源地址）。

在<description>中建议使用<![CDATA[ ]]>，所有在 CDATA 部件之间的文本都会被解析器忽略。

注意：CDATA 部件之间不能再包含 CDATA 部件（不能嵌套）。如果 CDATA 部件包含了字符“]]>”或者 CDATA，将很有可能出错哦。同样要注意在字符串“]]>”之间没有空格或者换行符。

14.2 SAX技术介绍

SAX，全称 Simple API for XML，既是指一种接口，也是指一个软件包。SAX 最初是由 David Megginson 采用 Java 语言开发，之后 SAX 很快在 Java 开发者中流行起来。San 现在负责管理其原始 API 的开发工作，这是一种公开的、开放源代码软件。不同于其他大多数 XML 标准的是，SAX 没有语言开发商必须遵守的标准 SAX 参考版本。因此，SAX 的不同实现可能采用区别很大的接口。在本节的内容中，简要介绍 SAX 技术的基本知识。

14.2.1 SAX的原理

作为接口，SAX 是事件驱动型 XML 解析的一个标准接口（Standard Interface）不会改变，已被 OASIS（Organization for the Advancement of Structured Information Standards）所采纳。作为软件包，SAX 最早的开发始于 1997 年 12 月，由一些在互联网上分散的程序员合作进行。



后来，参与开发的程序员越来越多，组成了互联网上的 XML-DEV 社区。5 个月以后，1998 年 5 月，SAX 1.0 版由 XML-DEV 正式发布。目前，最新的版本是 SAX 2.0。2.0 版本在多处与 1.0 版本不兼容，包括一些类和方法的名字。

SAX 的工作原理简单地说就是对文档进行顺序扫描，当扫描到文档（Document）开始与结束、元素（Element）开始与结束、文档（Document）结束等地方时通知事件处理函数，由事件处理函数做相应动作，然后继续同样的扫描，直至文档结束。

大多数 SAX 实现都会产生以下类型的事件。

- 在文档的开始和结束时触发文档处理事件。
- 在文档内每一 XML 元素接受解析的前后触发元素事件。
- 任何元数据通常都由单独的事件交付。
- 在处理文档的 DTD 或 Schema 时产生 DTD 或 Schema 事件。
- 产生错误事件用来通知主机应用程序解析错误。

14.2.2　基于对象和基于事件的接口

语法分析器有两类接口：基于对象的接口和基于事件的接口。DOM 是基于对象的语法分析器的标准的 API。作为基于对象的接口，DOM 通过在内存中显式地构建对象树来与应用程序通信。对象树是 XML 文件中元素树的精确映射。

DOM 易于学习和使用，因为它与基本 XML 文档紧密匹配。它对于以 XML 为中心的应用程序（例如，浏览器和编辑器）也是很理想的。以 XML 为中心的应用程序为了操纵 XML 文档而操纵 XML 文档。

然而，对于大多数应用程序，处理 XML 文档只是其众多任务中的一种。例如，记账软件包可能导入 XML 发票，但这不是其主要活动。计算账户余额、跟踪支出以及使付款与发票匹配才是主要活动。记账软件包可能已经具有一个数据结构（最有可能是数据库）。DOM 模型不太适合记账应用程序，因为在那种情况下，应用程序必须在内存中维护数据的两份副本（一个是 DOM 树，另一个是应用程序自己的结构）。至少，在内存维护两次数据会使效率下降。对于桌面应用程序来说，这可能不是主要问题，但是它可能导致服务器瘫痪。对于不以 XML 为中心的应用程序，SAX 是明智的选择。实际上，SAX 并不在内存中显式地构建文档树。它使应用程序能用最有效率的方法存储数据。

图 14-1 说明了应用程序如何在 XML 树及其自身数据结构之间进行映射。

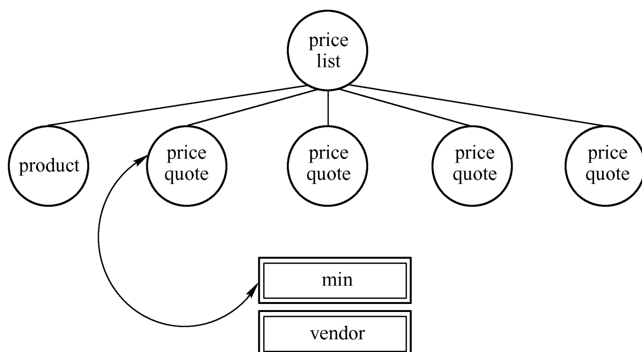


图 14-1　将 XML 结构映射成应用程序结构

SAX 是基于事件的接口，正如其名称所暗示的，基于事件的语法分析器将事件发送给应用程序。这些事件类似于用户界面事件，例如，浏览器中的 ONCLICK 事件或者 Java 中的 AWT/Swing 事件。

事件通知应用程序发生了某件事并需要应用程序作出反应。在浏览器中，通常为响应用户操作而生成事件：当用户单击按钮时，按钮产生一个 ONCLICK 事件。

在 XML 语法分析器中，事件与用户操作无关，而与正在读取的 XML 文档中的元素有关。有对于以下方面的事件。

- 元素开始和结束标记。
- 元素内容。
- 实体。
- 语法分析错误。

图 14-2 显示语法分析器在读取文档时如何生成事件。

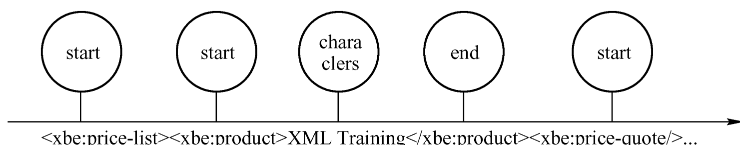


图 14-2　语法分析器生成事件

读者在此可能禁不住要问：为什么使用基于事件的接口？

这两种 API 中没有一种在本质上更好；它们适用于不同的需求。经验法则是在需要更多控制时使用 SAX；要增加方便性时，则使 DOM。例如，DOM 在脚本语言中很流行。

采用 SAX 的主要原因是效率。SAX 比 DOM 做的事要少，但提供了对语法分析器的更多控制。当然，如果语法分析器的工作减少，则意味着开发者有更多的工作要做。而且，正如已讨论的，SAX 比 DOM 消耗的资源要少，这只是因为它不需要构建文档树。在 XML 早期，DOM 得益于 W3C 批准的官方 API 这一身份。逐渐地，开发者选择了功能性而放弃了方便性，并转向了 SAX。

SAX 的主要限制是它无法向后浏览文档。实际上，激发一个事件后，语法分析器就将其忘记。应用程序必须显式地缓冲其感兴趣的事件。

14.2.3　常用的接口和类

SAX 将其事件分为如下几个接口。

- ContentHandler：定义与文档本身关联的事件（例如，开始和结束标记）。大多数应用程序都注册这些事件。
- DTDHandler：定义与 DTD 关联的事件。然而，它不定义足够的事件来完整地报告 DTD。如果需要对 DTD 进行语法分析，请使用可选的 DeclHandler。DeclHandler 是 SAX 的扩展，并且不是所有的语法分析器都支持它。
- EntityResolver：定义与装入实体关联的事件。只有少数几个应用程序注册这些事件。
- ErrorHandler：定义错误事件。许多应用程序注册这些事件以便用它们自己的方式报错。



为简化工作，SAX 在 DefaultHandler 类中提供了这些接口的默认实现。在大多数情况下，为应用程序扩展 DefaultHandler 并覆盖相关的方法要比直接实现一个接口更容易。

1．XMLReader

为注册事件处理器并启动语法分析器，应用程序使用 XMLReader 接口。如 parse()，这种 XMLReader 方法，启动语法分析：

```
parser.parse(args[0]);
```

XMLReader 的主要方法如下。

1）parse()：对 XML 文档进行语法分析。parse()有两个版本；一个接受文件名或 URL，另一个接受 InputSource 对象。

2）setContentHandler()、setDTDHandler()、setEntityResolver()和 setErrorHandler()：让应用程序注册事件处理器。

3）setFeature()和 setProperty()：控制语法分析器如何工作。它们采用一个特性或功能标识（一个类似于名称空间的 URI 和值）。功能采用 Boolean 值，而特性采用“对象”。

最常用的 XMLReaderFactory 功能如下。

1）http:// xml.org/sax/features/namespaces：所有 SAX 语法分析器都能识别它。如果将它设置为 true（默认值），则在调用 ContentHandler 的方法时，语法分析器将识别出名称空间并解析前缀。

2）http://xml.org/sax/features/validation：它是可选的。如果将它设置为 true，则验证语法分析器将验证该文档。非验证语法分析器忽略该功能。

2．XMLReaderFactory

XMLReaderFactory 创建语法分析器对象。它定义 createXMLReader()的两个版本：一个采用语法分析器的类名作为参数；另一个从 org.xml.sax.driver 系统特性中获得类名称。

对于 Xerces，类似 org.apache.xerces.parsers.SAXParser。应该使用 XMLReaderFactory，因为它易于切换至另一种 SAX 语法分析器。实际上，只需要更改一行然后重新编译。

```
XMLReaderparser=XMLReaderFactory.createXMLReader(
"org.apache.xerces.parsers.SAXParser");
```

为获得更大的灵活性，应用程序可以从命令行读取类名或使用不带参数的 createXMLReader()。因此，甚至可以不重新编译就更改语法分析器。

3．InputSource

InputSource 控制语法分析器如何读取文件，包括 XML 文档和实体。在大多数情况下，文档是从 URL 装入的。但是，有特殊需求的应用程序可以覆盖 InputSource。例如，这可以用来从数据库中装入文档。

4．ContentHandler

ContentHandler 是最常用的 SAX 接口，因为它定义 XML 文档的事件。ContentHandler 声明如下几个事件。

1）startDocument()/endDocument()：通知应用程序文档的开始或结束。

2）startElement()/endElement()：通知应用程序标记的开始或结束。属性作为 Attributes 参

数传递（请参阅下面一节“属性”）。即使只有一个标记，“空”元素（例如，<imghref="logo.gif"/>）也生成 startElement()和 endElement()。

3）startPrefixMapping()/endPrefixMapping()：通知应用程序名称空间作用域。用户几乎不需要该信息，因为当 http://xml.org/sax/features/namespaces 为 true 时，语法分析器已经解析了名称空间。

4）当语法分析器在元素中发现文本（已经过语法分析的字符数据）时，characters()/ignorableWhitespace()会通知应用程序。要知道，语法分析器负责将文本分配到几个事件（更好地管理其缓冲区）。ignorableWhitespace 事件用于由 XML 标准定义的可忽略空格。

5）processingInstruction()：将处理指令通知应用程序。

6）skippedEntity()：通知应用程序已经跳过了一个实体（即，当语法分析器未在 DTD/schema 中发现实体声明时）。

7）setDocumentLocator()：将 Locator 对象传递到应用程序；请参阅后面的 Locator 一节。请注意，不需要 SAX 语法分析器提供 Locator，但是如果它提供了，则必须在任何其他事件之前激活该事件。

5．属性

在 startElement()事件中，应用程序在 Attributes 参数中接收属性列表。

```
Stringattribute=attributes.getValue("","price");
```

Attributes 定义下列方法。

- getValue(i)/getValue(qName)/getValue(uri,localName)返回第 i 个属性值或给定名称的属性值。
- getLength()返回属性数目。
- getQName(i)/getLocalName(i)/getURI(i)返回限定名（带前缀）、本地名（不带前缀）和第 i 个属性的名称空间 URI。
- getType(i)/getType(qName)/getType(uri,localName)返回第 i 个属性的类型或者给定名称的属性类型。类型为字符串，即在 DTD 所使用的“CDATA”、“ID”、“IDREF”、“IDREFS”、“NMTOKEN”、“NMTOKENS”、“ENTITY”、“ENTITIES”或“NOTATION”

注意：Attributes 参数仅在 startElement()事件期间可用。如果在事件之间需要它，则用 AttributesImpl 复制一个。

6．定位器

Locator 为应用程序提供行和列的位置。不需要语法分析器来提供 Locator 对象。Locator 定义下列方法。

- getColumnNumber()返回当前事件结束时所在的那一列。在 endElement()事件中，它将返回结束标记所在的最后一列。
- getLineNumber()返回当前事件结束时所在的行。在 endElement()事件中，它将返回结束标记所在的行。
- getPublicId()返回当前文档事件的公共标识。
- getSystemId()返回当前文档事件的系统标识。



7．DTDHandler

DTDHandler 声明两个与 DTD 语法分析器相关的事件。具体如下。

- notationDecl()通知应用程序已经声明了一个标记。
- nparsedEntityDecl()通知应用程序已经发现了一个未经过语法分析的实体声明。

8．EntityResolver

EntityResolver 接口仅定义一个事件 resolveEntity()，它返回 InputSource（在另一章讨论）。因为 SAX 语法分析器已经可以解析大多数 URL，所以很少应用程序实现 EntityResolver。例外情况是目录文件（在另一章中讨论），它将公共标识解析成系统标识。如果在应用程序中需要目录文件，请下载 NormanWalsh 的目录软件包（请参阅参考资料）。

9．ErrorHandler

ErrorHandler 接口定义错误事件。处理这些事件的应用程序可以提供定制错误处理。安装了定制错误处理器后，语法分析器不再抛出异常，抛出异常是事件处理器的责任。接口定义了错误的 3 个级别。

- warning()：警示那些不是由 XML 规范定义的错误。例如，当没有 XML 声明时，某些语法分析器发出警告。它不是错误（因为声明是可选的），但是它可能值得注意。
- error()：警示那些由 XML 规范定义的错误。
- fatalError()：警示那些由 XML 规范定义的致命错误。

10．SAXException

SAX 定义的大多数方法都可以抛出 SAXException。当对 XML 文档进行语法分析时，SAXException 通知一个错误。错误可以是语法分析错误也可以是事件处理器中的错误。要报告来自事件处理器的其他异常，可以将异常封装在 SAXException 中。

14.3　实现流程

本项目实例的功能是，在手机中显示指定 RSS 的信息，即设置显示网易博客 http://feed.feedsky.com/asdfg343442用户的日志信息。

本项目的具体实现流程如图 14-3 所示。

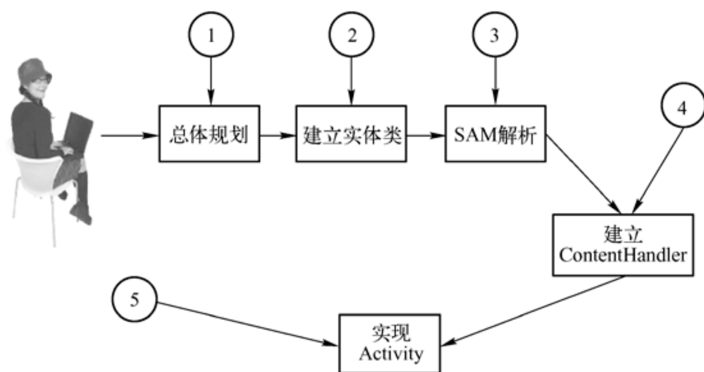


图 14-3 实现流程图

14.4 具体实现

从本节内容开始，将着重介绍本项目的具体实现过程。详细讲解各个代码的具体实现过程，并讲解其中的技巧和要点，使读者的水平更上一层楼。

14.4.1 建立实体类

一个 RSS 文件可以被认为是由一个 RSS 的一些描述性信息和里面的 item 元素组成的，例如关于 RSS 的描述性信息如下。

- title：标题信息。
- link：链接信息。
- description：描述信息。

item 里面的信息如下。

- title：标题信息。
- link：链接信息。
- description：描述信息。
- pubDate：发布的日期。

在本项目实例中需要建立如下两个实体类。

- RSSFeed：用于和一个 RSS 的完整 XML 文件相对应。
- RSSItem：用于和一个 RSS 中 Item 标签相对应。

在解析 RSS 文件时，可以将文件里的信息解析出来放到实体类里面，这样就可以直接操作该实体类了。下面开始讲解上述两个实体类的具体实现过程。

1．RSSFeed类

RSSFeed 类的功能是建立和一个完整 XML 文件的对应，其中方法 addItem 用于将一个 RSSItem 添加到 RSSFeed 类里面；方法 getAllItemsForListView 负责从 RSSFeed 类里面生成 ListView 列表所需要的数据。RSSFeed 类的具体实现代码如下所示。

```
package com.rss_reader.data;
import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;
import java.util.Vector;

public class RSSFeed
{
	private String title = null;
	private String pubdate = null;
	private int itemcount = 0;
	private List<RSSItem> itemlist;
```



```
 public RSSFeed()
{
        itemlist = new Vector(0);
}
public int addItem(RSSItem item)
{
        itemlist.add(item);
        itemcount++;
        return itemcount;
}
public RSSItem getItem(int location)
{
        return itemlist.get(location);
}
public List getAllItems()
{
        return itemlist;
}
public List getAllItemsForListView(){
        List<Map<String, Object>> data = new ArrayList<Map<String, Object>>();
        int size = itemlist.size();
        for(int i=0;i<size;i++){
                HashMap<String, Object>     item = new HashMap<String, Object>();
                item.put(RSSItem.TITLE, itemlist.get(i).getTitle());
                item.put(RSSItem.PUBDATE, itemlist.get(i).getPubDate());
                data.add(item);
        }
        return data;
}
int getItemCount()
{
        return itemcount;
}
public void setTitle(String title)
{
        this.title = title;
}
public void setPubDate(String pubdate)
{
        this.pubdate = pubdate;
}
public String getTitle()
{
        return title;
}
```

```
        public String getPubDate()
        {
                return pubdate;
        }




}
```

2．RSSItem类

RSSFeed 类的功能是，用于和一个 RSS 中 Item 标签相对应，它里面的属性和 item 里面的属性一样。RSSItem 类的具体实现代码如下所示。

```
package com.rss_reader.data;

public class RSSItem
{
        public static final String TITLE="title";
        public static final String PUBDATE="pubdate";
        private String title = null;
        private String description = null;
        private String link = null;
        private String category = null;
        private String pubdate = null;


        public RSSItem()
        {
        }
        public void setTitle(String title)
        {
                this.title = title;
        }
        public      void setDescription(String description)
        {
                this.description = description;
        }
        public      void setLink(String link)
        {
                this.link = link;
        }
        public      void setCategory(String category)
        {
                this.category = category;
        }
        public      void setPubDate(String pubdate)
```



```
        {
            this.pubdate = pubdate;
        }
        public String getTitle()
        {
            return title;
        }
        public    String getDescription()
        {
            return description;
        }
        public    String getLink()
        {
            return link;
        }
        public String getCategory()
        {
            return category;
        }
        public String getPubDate()
        {
            return pubdate;
        }
        public String toString()
        {
            if (title.length() > 20)
            {
                return title.substring(0, 42) + "...";
            }
            return title;
        }
    }
```

14.4.2　主程序文件ActivityMain.java

主程序文件 ActivityMain.java 是本项目的入口，在此 Activity 中得到了服务器端的 RSSFeed，经过解析后将里面的内容以 ListView 的形式显示出来。下面开始讲解其具体实现流程。

1）先 import 相关 class，然后设置目标 RSS 的源地址为http://feed.feedsky.com/asdfg343442，最后通过 showListView()方法将获取的 RSS 信息以列表形式显示出来。具体代码如下所示。

```
    package com.rss_reader;

    import java.net.URL;
```

```
import javax.xml.parsers.SAXParser;
import javax.xml.parsers.SAXParserFactory;

import org.xml.sax.InputSource;
import org.xml.sax.XMLReader;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.AdapterView.OnItemClickListener;

import com.rss_reader.data.RSSFeed;
import com.rss_reader.data.RSSItem;
import com.rss_reader.sax.RSSHandler;

public class ActivityMain extends Activity implements OnItemClickListener
{
//      public final String RSS_URL = "http://rubyjin.cn/blog/rss";

                public final String RSS_URL = "http://feed.feedsky.com/asdfg343442";

        public final String tag = "RSSReader";
        private RSSFeed feed = null;

        /** Called when the activity is first created. */

       public void onCreate(Bundle icicle) {
              super.onCreate(icicle);
              setContentView(R.layout.main);
              feed = getFeed(RSS_URL);
              showListView();

       }
```

2）定义方法 getFeed(String urlString)，用于得到一个 RSSFeed，即从服务器端请求了 RSS Feed，并进行了解析，将解析后的内容都放在 RSSFeed 的一个实例里面。上述解析过程是通过 SAX 实现的，具体流程如下。

第 1 步：新建工厂类 SAXParserFactory。

第 2 步：工厂类产出一个 SAX 解析类 SAXParser。

第 3 步：从 SAXParser 中得到一个 XMLReader 实例，XMLReader 是一个接口，此接口



中定义了一些解析 XML 的回调函数。

第 4 步：把编写的 Handler 注册到 XMLReader 中去。

第 5 步：将一个 XML 文档或资源变成一个 Java 可以处理的 InputStream 流后，解析工作开始。

具体代码如下所示。

```
private RSSFeed getFeed(String urlString)
{
    try
    {
        URL url = new URL(urlString);
        /*新建工厂类 SAXParserFactory*/
        SAXParserFactory factory = SAXParserFactory.newInstance();
        /*工厂类产出一个 SAX 解析类 SAXParser */
        SAXParser parser = factory.newSAXParser();
        /*从 SAXParser 中得到一个 XMLReader 实例*/
        XMLReader xmlreader = parser.getXMLReader();
        /*把编写的 Handler 注册到 XMLReader 中  */
        RSSHandler rssHandler = new RSSHandler();
        xmlreader.setContentHandler(rssHandler);

        /*将一个 XML 文档或资源变成一个 Java 可以处理的 InputStream 流后，解析工作开始.*/
        InputSource is = new InputSource(url.openStream());

        xmlreader.parse(is);

        return rssHandler.getFeed();
    }
    catch (Exception ee)
    {

        return null;
    }
}
```

3）定义方法 showListView()，用于列表显示获取的 RSS，这样 ListView 和一个 SimpleAdapter 实现了绑定。具体代码如下所示。

```
private void showListView()
{
    ListView itemlist = (ListView) findViewById(R.id.itemlist);
    if (feed == null)
    {
        setTitle("访问的 RSS 无效");
        return;
    }
```

```
        SimpleAdapter adapter = new SimpleAdapter(this, feed.getAllItemsForListView(),
                android.R.layout.simple_list_item_2, new String[] { RSSItem.TITLE,RSSItem.PUBDATE },
                new int[] { android.R.id.text1 , android.R.id.text2});
        itemlist.setAdapter(adapter);
        itemlist.setOnItemClickListener(this);
        itemlist.setSelection(0);

    }
```

4）定义方法 onItemClick，用于处理列表的单击事件，当单击后会显示此 RSS 信息的链接地址，用户单击后可以通过浏览器来到目标地址。具体代码如下所示。

```
public void onItemClick(AdapterView parent, View v, int position, long id)
{
    Intent itemintent = new Intent(this,ActivityShowDescription.class);

    Bundle b = new Bundle();
    b.putString("title", feed.getItem(position).getTitle());
    b.putString("description", feed.getItem(position).getDescription());
    b.putString("link", feed.getItem(position).getLink());
    b.putString("pubdate", feed.getItem(position).getPubDate());

 itemintent.putExtra("android.intent.extra.rssItem", b);
      startActivityForResult(itemintent, 0);
}
```

14.4.3 实现ContentHandler

ContentHandler 是一个特殊的 SAX 接口，位于 org.xml.sax.ContentHandler。在解析 XML 时，大多数步骤都是固定不变的，但是关于 ContentHandler 的实现确实不同的。实现 ContentHandler 是解析 XML 中最重要、最关键的步骤之一，下面将开始讲解其具体实现流程。

1）声明 RSSHandler 类，声明继承与 DefaultHandler 的类。DefaultHandler 类是一个基类，此类里面最简单的实现了一个 ContentHandler，只需在里面重写里面的重要方法即可。具体代码如下所示。

```
package com.rss_reader.sax;


import org.xml.sax.Attributes;
import org.xml.sax.SAXException;
import org.xml.sax.helpers.DefaultHandler;

import android.util.Log;


import com.rss_reader.data.RSSFeed;
import com.rss_reader.data.RSSItem;
```



```
public class RSSHandler extends DefaultHandler
{

    RSSFeed rssFeed;
    RSSItem rssItem;
    String lastElementName = "";
    final int RSS_TITLE = 1;
    final int RSS_LINK = 2;
    final int RSS_DESCRIPTION = 3;
    final int RSS_CATEGORY = 4;
    final int RSS_PUBDATE = 5;

    int currentstate = 0;

    public RSSHandler()
    {
    }

    public RSSFeed getFeed()
    {
        return rssFeed;
    }
```

2）分别重写 startDocument()和 endDocument()，通常将正式解析前的初始化工作放到 startDocument()中，将一些收尾性工作放到 endDocument()中。具体代码如下所示。

```
public void startDocument() throws SAXException
{
    rssFeed = new RSSFeed();
    rssItem = new RSSItem();

}
public void endDocument() throws SAXException
{
}
```

3）重写 startElement，当 XML 解析器遇到 XML 文档流里面的 tag 时，将会调用此函数。在此函数内部通常是通过参数 localName 进行判断并进行一些操作处理的。具体代码如下所示。

```
public void startElement(String namespaceURI, String localName,String qName, Attributes atts) throws
SAXException
{
    if (localName.equals("channel"))
    {
        currentstate = 0;
```

```
                return;
            }
            if (localName.equals("item"))
            {
                rssItem = new RSSItem();
                return;
            }
            if (localName.equals("title"))
            {
                currentstate = RSS_TITLE;
                return;
            }
            if (localName.equals("description"))
            {
                currentstate = RSS_DESCRIPTION;
                return;
            }
            if (localName.equals("link"))
            {
                currentstate = RSS_LINK;
                return;
            }
            if (localName.equals("category"))
            {
                currentstate = RSS_CATEGORY;
                return;
            }
            if (localName.equals("pubDate"))
            {
                currentstate = RSS_PUBDATE;
                return;
            }

            currentstate = 0;
        }
```

4）重写 endElement，此方法和 startElement 方法相对应，当解析 tag 完毕后执行此方法。如果解析一个 item 节点结束，就将 RSSItem 添加到 RSSFeed 中去。具体代码如下所示。

```
        public void endElement(String namespaceURI, String localName, String qName) throws SAXException
        {

            //如果解析一个 item 节点结束，就将 rssItem 添加到 rssFeed 中。
            if (localName.equals("item"))
            {
```



```
                        rssFeed.addItem(rssItem);
                        return;

                }
        }
```

5）重写 characters，此方法是一个回调方法，当解析完 startElement 方法后，解析完节点内容后会执行此方法，并且参数 ch[]就是节点的内容。具体代码如下所示。

```
        public void characters(char ch[], int start, int length)
        {
                String theString = new String(ch,start,length);

                switch (currentstate)
                {
                        case RSS_TITLE:
                                rssItem.setTitle(theString);
                                currentstate = 0;
                                break;
                        case RSS_LINK:
                                rssItem.setLink(theString);
                                currentstate = 0;
                                break;
                        case RSS_DESCRIPTION:
                                rssItem.setDescription(theString);
                                currentstate = 0;
                                break;
                        case RSS_CATEGORY:
                                rssItem.setCategory(theString);
                                currentstate = 0;
                                break;
                        case RSS_PUBDATE:
                                rssItem.setPubDate(theString);
                                currentstate = 0;
                                break;
                        default:
                                return;
                }

        }
    }
```

14.4.4　主程序文件ActivityShowDescription.java

主程序文件 ActivityShowDescription.java 的功能是，显示某列表信息的详细信息。当单击列表中的某一项后，会进入到此界面。如果程序出错，则 content 显示出错提示；运行正确则

在 content 中分别显示 title、pubdate 和 description。具体代码如下所示。

```
package com.rss_reader;

import android.app.Activity;
import android.os.Bundle;
import android.widget.Button;
import android.widget.TextView;
import android.content.Intent;
import android.view.*;

public class ActivityShowDescription extends Activity {
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.showdescription);
        String content = null;
        Intent startingIntent = getIntent();

        if (startingIntent != null) {
            Bundle bundle = startingIntent
                    .getBundleExtra("android.intent.extra.rssItem");
            if (bundle == null) {
                content = "不好意思程序出错啦";
            } else {
                content = bundle.getString("title") + "\n\n"
                        + bundle.getString("pubdate") + "\n\n"
                        + bundle.getString("description").replace('\n', ' ')
                        + "\n\n 详细信息请访问以下网址:\n" + bundle.getString("link");
            }
        } else {
            content = "不好意思程序出错啦";

        }

        TextView textView = (TextView) findViewById(R.id.content);
        textView.setText(content);

        Button backbutton = (Button) findViewById(R.id.back);

        backbutton.setOnClickListener(new Button.OnClickListener() {
            public void onClick(View v) {
                finish();
            }
        });
    }
}
```



14.4.5　主布局文件main.xml

主布局文件 main.xml 用于定义系统初始主界面，即列表显示获取的 RSS 信息。具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<ListView
     android:layout_width="fill_parent"
     android:layout_height="fill_parent"
     android:id="@+id/itemlist"

     />
</LinearLayout>
```

14.4.6　详情主布局文件showdescription.xml

当用户单击列表信息后，会进入信息详情界面，此界面是由布局文件 showdescription.xml 定义的。具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:autoLink="all"
    android:text=""
    android:id="@+id/content"
    android:layout_weight="1.0"
    />
<Button
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="返回"
     android:id="@+id/back"
     />

</LinearLayout>
```

至此，整个实例介绍完毕。运行后将获取指定 RSS 中的信息，如图 14-4 所示。单击某条信息后会显示此信息的相关描述性信息，如图 14-5 所示。



图 14-4　初始效果

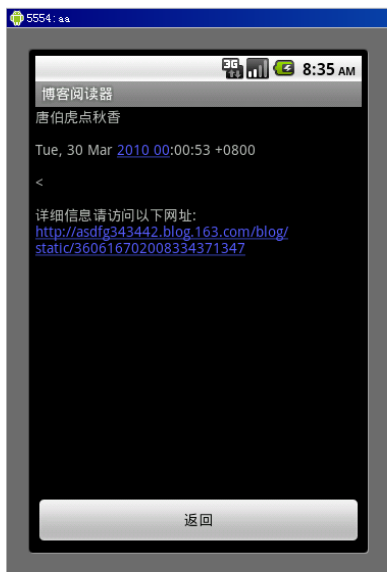


图 14-5　详情界面

单击图 14-5 中间的链接后，能够显示此条 RSS 的详细信息，如图 14-6 所示。

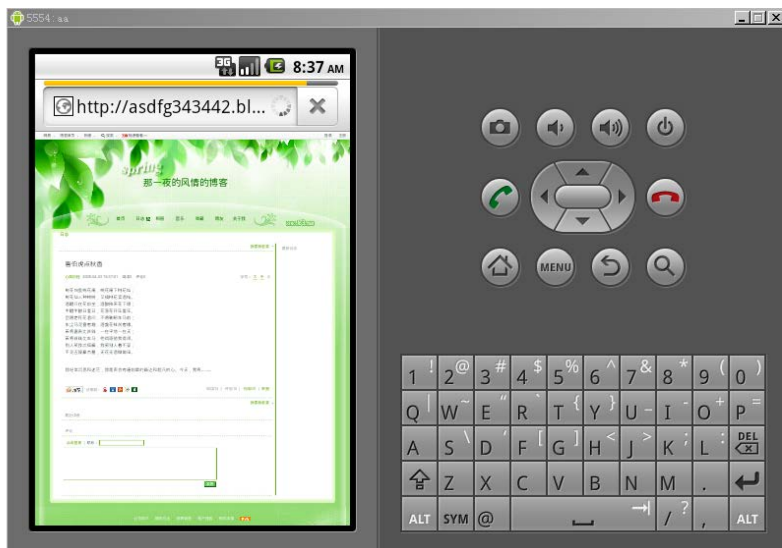


图 14-6　详细信息

本实例默认显示的是博客 http://asdfg343442.blog.163.com/中的信息。读者也可以指定显示其他 RSS 信息，在使用时可以登录 http://www.feedsky.com/来设置不同的 RSS 订阅。具体设置流程如下。

1）来到 http://www.feedsky.com/主界面，如图 14-7 所示。



图 14-7　feedsky.com 主界面

2）在图 14-7 顶部的文本框中输入要显示信息的博客地址、Feed 地址或 QQ 号码，然后单击“下一步”按钮，如图 14-8 所示。

图 14-8　输入设置的博客、QQ 或 Feed 地址

3）在弹出界面中分别输入“名称”、“描述”和“tag”，并设定永久性 Feed 地址，如图 14-9 所示。

图 14-9　添加 Feed 界面



图 14-9 中的永久 Feed 地址就是 RSS 的源地址，这样就可以将此地址添加到实例中，从而显示此地址的 RSS 资源信息，也就是显示博客 http://asdfg343442.blog.163.com/中的信息。

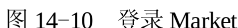
14.5 打包、签名和发布

当一个 Android 项目开发完毕后，需要打包和签名处理，这样才能放到手机中使用，当然也可以发布到 Market 上去赚钱。下面开始讲解打包、签名、发布 Android 程序的具体过程。

14.5.1 申请会员

去 Market 申请成为会员，具体流程如下。

1）登录http://market.android/publish/signup，如图 14-10 所示。


图 14-10 登录 Market

2）单击链接“Create an account now”，来到注册页面，如图 14-11 所示。

图 14-11 注册界面



3）单击同意协议后来到下一步页面，在此输入手机号码，如图 14-12 所示。

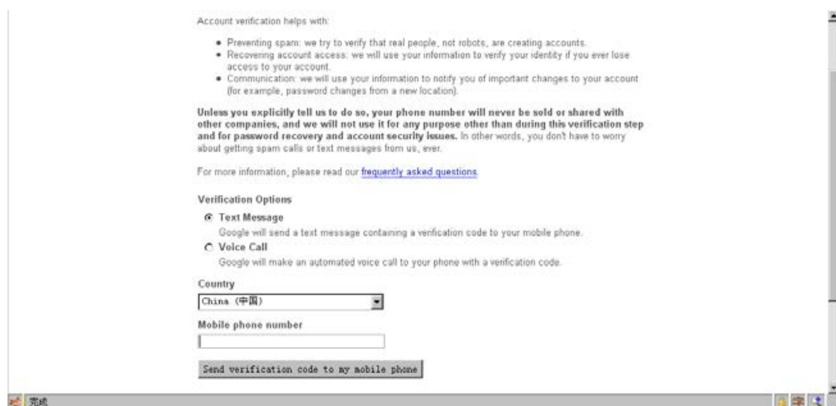


图 14-12　输入手机号码

4）在新界面中输入手机获取的验证码，如图 14-13 所示。

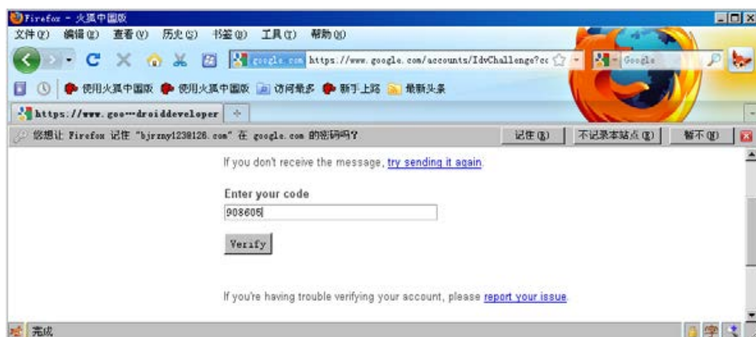


图 14-13　输入验证码

5）验证通过后，在新界面中继续输入信息，如图 14-14 所示。



图 14-14　输入信息

6）单击“Continue »”按钮后，提示需要花费 25 美元，支付后才能成为正式会员，如图 14-15 所示。



图 14-15　需要支付界面

7）单击 Google Checkout（Fast checkout through Google）按钮来到支付界面，如图 14-16 所示。

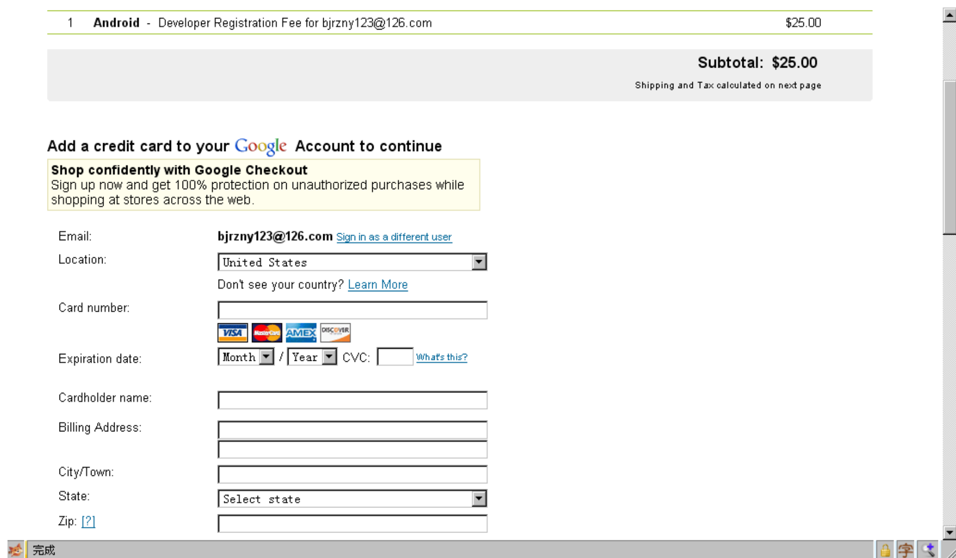


图 14-16　支付界面

在此输入信用卡信息，完成支付后即可成为正式会员。

14.5.2　生成签名文件

Android 程序的签名和 Symbian 类都可以自签名(Self-signed)，但是在 Android 平台中证书初期还显得形同虚设，平时开发时通过 ADB 接口上传的程序会自动被签有 Debug 权限的程序。需要签名验证在上传程序到 Android Market 上时大家都已经发现这个问题了。



Androi 签名文件的制作方法有两种。

第一种：命令行生成。

具体流程如下。

1）输入如下命令。

```
keytool -genkey -alias android123.keystore -keyalg RSA -validity 20000 -keystore android123.keystore
```

然后一次提示用户输入如下信息。

输入 keystore 密码：[密码不回显]。

再次输入新密码：[密码不回显]。

您的名字与姓氏是什么？

[Unknown]：　android123

您的组织单位名称是什么？

[Unknown]：　www.android123.com.cn

您的组织名称是什么？

[Unknown]：　www.android123.com.cn

您的组织名称是什么？

[Unknown]：　www.android123.com.cn

您所在的城市或区域名称是什么？

[Unknown]：　New York

您所在的州或省份名称是什么？

[Unknown]：　New York

该单位的两字母国家代码是什么？

[Unknown]：　CN

CN=android123, OU=www.android123.com.cn, O=www.android123.com.cn, L=New York, ST =New York, C=CN　正确吗？

[否]：Y

输入<android123.keystore>的主密码（如果和 keystore 密码相同，按回车）。

其中参数–validity 为证书有效天数，这里写得大些，还有在输入密码时没有回显，只管输入就可以了，一般位数建议使用 20 位，最后需要记下来后面还要用。接下来就可以为 apk 文件签名了。

2）执行。

```
jarsigner -verbose -keystore android123.keystore -signedjar android123_signed.apk android123.apk android123.keystore
```

这样就可以生成签名的 apk 文件，假设输入文件 android123.apk，则最终生成 android123_signed.apk 为 Android 签名后的 APK 执行文件。

keytool 用法和 jarsigner 用法总结如下。

（1）keytool 用法

-certreq　　[-v] [-protected]

```
                 [-alias <别名>] [-sigalg <sigalg>]
                 [-file <csr_file>] [-keypass <密钥库口令>]
                 [-keystore <密钥库>] [-storepass <存储库口令>]
                 [-storetype <存储类型>] [-providername <名称>]
                 [-providerclass <提供方类名称> [-providerarg <参数>]] ...
                 [-providerpath <路径列表>]

-changealias [-v] [-protected] -alias <别名> -destalias <目标别名>
                 [-keypass <密钥库口令>]
                 [-keystore <密钥库>] [-storepass <存储库口令>]
                 [-storetype <存储类型>] [-providername <名称>]
                 [-providerclass <提供方类名称> [-providerarg <参数>]] ...
                 [-providerpath <路径列表>]

-delete          [-v] [-protected] -alias <别名>
                 [-keystore <密钥库>] [-storepass <存储库口令>]
                 [-storetype <存储类型>] [-providername <名称>]
                 [-providerclass <提供方类名称> [-providerarg <参数>]] ...
                 [-providerpath <路径列表>]

-exportcert [-v] [-rfc] [-protected]
                 [-alias <别名>] [-file <认证文件>]
                 [-keystore <密钥库>] [-storepass <存储库口令>]
                 [-storetype <存储类型>] [-providername <名称>]
                 [-providerclass <提供方类名称> [-providerarg <参数>]] ...
                 [-providerpath <路径列表>]

-genkeypair [-v] [-protected]
                 [-alias <别名>]
                 [-keyalg <keyalg>] [-keysize <密钥大小>]
                 [-sigalg <sigalg>] [-dname <dname>]
                 [-validity <valDays>] [-keypass <密钥库口令>]
                 [-keystore <密钥库>] [-storepass <存储库口令>]
                 [-storetype <存储类型>] [-providername <名称>]
                 [-providerclass <提供方类名称> [-providerarg <参数>]] ...
                 [-providerpath <路径列表>]

-genseckey       [-v] [-protected]
                 [-alias <别名>] [-keypass <密钥库口令>]
```



```
                 [-keyalg <keyalg>] [-keysize <密钥大小>]
                 [-keystore <密钥库>] [-storepass <存储库口令>]
                 [-storetype <存储类型>] [-providername <名称>]
                 [-providerclass <提供方类名称> [-providerarg <参数>]] ...
                 [-providerpath <路径列表>]

-help
-importcert [-v] [-noprompt] [-trustcacerts] [-protected]
                 [-alias <别名>]
                 [-file <认证文件>] [-keypass <密钥库口令>]
                 [-keystore <密钥库>] [-storepass <存储库口令>]
                 [-storetype <存储类型>] [-providername <名称>]
                 [-providerclass <提供方类名称> [-providerarg <参数>]] ...
                 [-providerpath <路径列表>]

-importkeystore [-v]
                 [-srckeystore <源密钥库>] [-destkeystore <目标密钥库>]
                 [-srcstoretype <源存储类型>] [-deststoretype <目标存储类型>]
                 [-srcstorepass <源存储库口令>] [-deststorepass <目标存储库口令>]
                 [-srcprotected] [-destprotected]
                 [-srcprovidername <源提供方名称>]
                 [-destprovidername <目标提供方名称>]
                 [-srcalias <源别名> [-destalias <目标别名>]
                   [-srckeypass <源密钥库口令>] [-destkeypass <目标密钥库口令>]]
                 [-noprompt]
                 [-providerclass <提供方类名称> [-providerarg <参数>]] ...
                 [-providerpath <路径列表>]

-keypasswd      [-v] [-alias <别名>]
                 [-keypass <旧密钥库口令>] [-new <新密钥库口令>]
                 [-keystore <密钥库>] [-storepass <存储库口令>]
                 [-storetype <存储类型>] [-providername <名称>]
                 [-providerclass <提供方类名称> [-providerarg <参数>]] ...
                 [-providerpath <路径列表>]

-list           [-v | -rfc] [-protected]
                 [-alias <别名>]
                 [-keystore <密钥库>] [-storepass <存储库口令>]
                 [-storetype <存储类型>] [-providername <名称>]
```

```
                [-providerclass <提供方类名称> [-providerarg <参数>]] ...
                [-providerpath <路径列表>]

-printcert     [-v] [-file <认证文件>]

-storepasswd [-v] [-new <新存储库口令>]
                [-keystore <密钥库>] [-storepass <存储库口令>]
                [-storetype <存储类型>] [-providername <名称>]
                [-providerclass <提供方类名称> [-providerarg <参数>]] ...
                [-providerpath <路径列表>]
```

（2）jarsigner 用法

[选项] jar 文件别名

jarsigner -verify [选项] jar 文件

[-keystore <url>]	密钥库位置
[-storepass <口令>]	用于密钥库完整性的口令
[-storetype <类型>]	密钥库类型
[-keypass <口令>]	专用密钥的口令（如果不同）
[-sigfile <文件>]	.SF/.DSA 文件的名称
[-signedjar <文件>]	已签名的 JAR 文件的名称
[-digestalg <算法>]	摘要算法的名称
[-sigalg <算法>]	签名算法的名称
[-verify]	验证已签名的 JAR 文件
[-verbose]	签名/验证时输出详细信息
[-certs]	输出详细信息和验证时显示证书
[-tsa <url>]	时间戳机构的位置
[-tsacert <别名>]	时间戳机构的公共密钥证书
[-altsigner <类>]	替代的签名机制的类名
[-altsignerpath <路径列表>]	替代的签名机制的位置
[-internalsf]	在签名块内包含 .SF 文件
[-sectionsonly]	不计算整个清单的散列
[-protected]	密钥库已保护验证路径
[-providerName <名称>]	提供者名称
[-providerClass <类>	加密服务提供者的名称
[-providerArg <参数>]] ...	主类文件和构造函数参数

第二种：Eclipse 的 ADT 生成。

实际上，使用 Eclipse 可以更加直观、方便的生成签名文件，具体流程如下。



1）右键单击 Eclipse 项目名，依次选择“Android Tools”→“Export Signed Application Package...”，如图 14-17 所示。

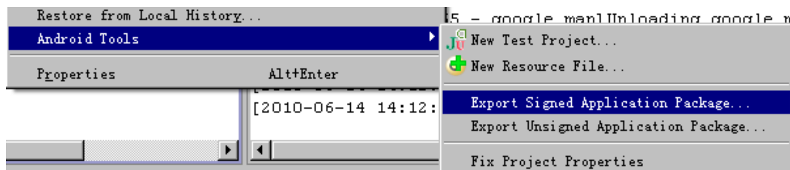
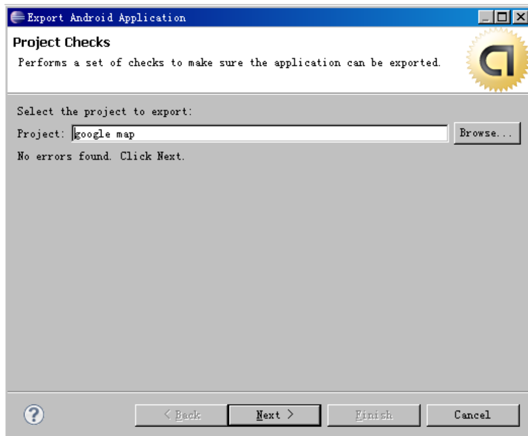


图 14-17　选择导出

2）在弹出界面中选择要导出的项目，如图 14-18 所示。

3）单击“Next”按钮，在弹出界面中选择 Create new keystore，然后分别输入文件名和密码，如图 14-19 所示。



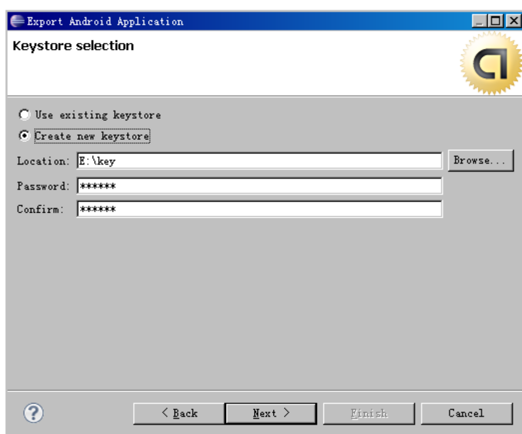
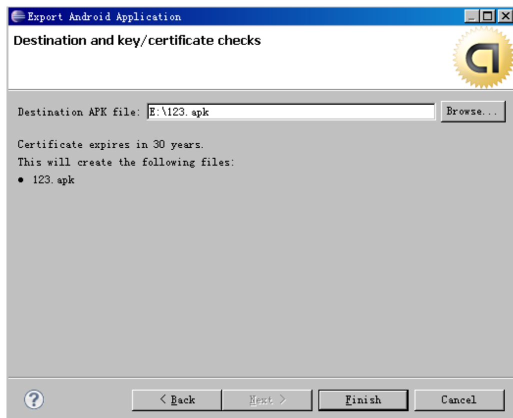


图 14-18　选择要导出的项目　　　　图 14-19　文件名和密码

4）单击“Next”按钮，在弹出界面中输入签名文件路径，如图 14-20 所示。

5）单击“Next”按钮，在弹出界面中一次输入签名文件的相关信息，如图 14-21 所示。



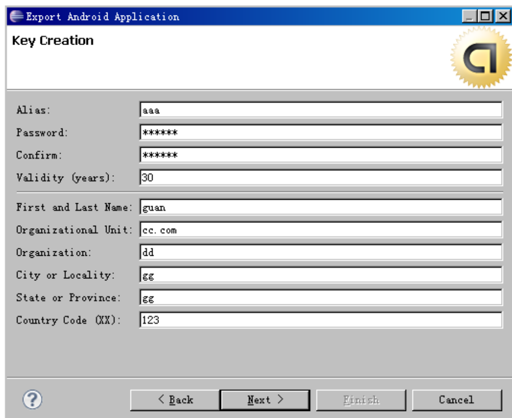


图 14-20　输入信息　　　　图 14-21　输入信息

6）单击“Next”按钮后完成签名文件的创建。

14.5.3 使用签名文件

生成签名文件后，就可以使用它了，在此也有两种方式。

第一种：命令行。

1）假设生成的签名文件是 ChangeBackgroundWidget.apk，则最终生成 ChangeBackgroundWidget_signed.apk 为 Android 签名后的 APK 执行文件。

输入以下命令行。

```
jarsigner -verbose -keystore ChangeBackgroundWidget.keystore -signedjar ChangeBackgroundWidget_signed.apk ChangeBackgroundWidget.apk ChangeBackgroundWidget.keystore
```

上面命令中间不换行。

2）按“Enter”键，根据提示输入密钥库的口令短语（即密码），详细信息如下。

输入密钥库的口令短语。

```
正在添加：  META-INF/MANIFEST.MF
正在添加：  META-INF/CHANGEBA.SF
正在添加：META-INF/CHANGEBA.RSA
正在签名：res/drawable/icon.png
正在签名：res/drawable/icon_audio.png
正在签名：res/drawable/icon_exit.png
正在签名：res/drawable/icon_folder.png
正在签名：res/drawable/icon_home.png
正在签名：res/drawable/icon_img.png
正在签名：res/drawable/icon_left.png
正在签名：res/drawable/icon_mantou.png
正在签名：res/drawable/icon_other.png
正在签名：res/drawable/icon_pause.png
正在签名：res/drawable/icon_play.png
正在签名：res/drawable/icon_return.png
正在签名：res/drawable/icon_right.png
正在签名：res/drawable/icon_set.png
正在签名：res/drawable/icon_text.png
正在签名：res/drawable/icon_xin.png
正在签名：res/layout/fileitem.xml
正在签名：res/layout/filelist.xml
正在签名：res/layout/main.xml
正在签名：res/layout/widget.xml
正在签名：res/xml/widget_info.xml
正在签名：AndroidManifest.xml
正在签名：resources.arsc
正在签名：classes.dex
```

通过上述过程处理后，即可将未签名文件 ChangeBackgroundWidget.apk 签名为



ChangeBackgroundWidget_signed.apk。

在上述方式中，读者可能会遇到以下问题。

问题一：jarsigner：无法打开 jar 文件 ChangeBackgroundWidget.apk。

解决方法：将要进行签名的 APK 放到对应的文件下，把要签名的 ChangeBackground Widget.apk 放到 JDK 的 bin 文件里。

问题二：jarsigner 无法对 jar 进行签名：java.util.zip.ZipException: invalid entry comp ressed size (expected 1598 but got 1622 bytes)。

方法一：Android 开发网提示这些问题主要是由于资源文件造成的，对于 Android 开发来说应该检查 res 文件夹中的文件，逐个排查。这个问题可以通过升级系统的 JDK 和 JRE 版本来解决。

方法二：这是因为默认给 apk 做了 debug 签名，所以无法做新的签名，这时就必须点右键→“Android Tools”→“Export Unsigned Application Package.”。

或者从 AndroidManifest.xml 的 Exporting 上也是一样的。

然后再基于这个导出的 unsigned apk 做签名，导出的时候最好将其目录选在之前产生 keystore 的那个目录下，这样操作起来就方便了。

第二种：Eclipse 的 ADT 生成。

实际上，使用 Eclipse 可以更加直观、方便的生成签名文件，具体流程如下。

1）右键单击 Eclipse 项目名，依次选择“Android Tools”→“Export Signed Application Package…”，如图 14-22 所示。

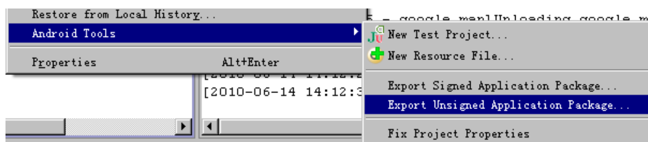


图 14-22　Export Unsigned Application Package

2）在弹出界面中选择项目，如图 14-23 所示。

3）单击“Next”按钮，在弹出界面中选择“Use existing keystore”，并输入文件的密码，如图 14-24 所示。

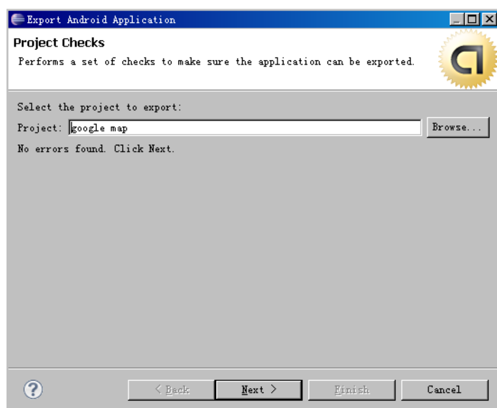


图 14-23　选择项目

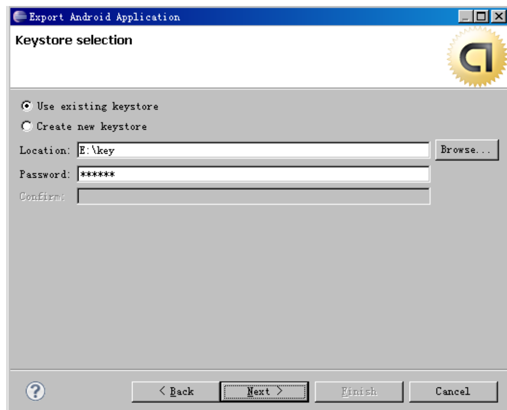


图 14-24　输入密码

4）单击“Next”按钮，输入原来签名文件的资料和密码，按照默认提示完成签名。

14.5.4 发布

发布的过程比较简单，来到 Market，登录个人中心，上传签名后的文件即可，具体操作流程在 Market 站点上有详细介绍，在此将不做详细介绍。

第 15 章　通用手机助手

在手机应用中，为了便于用户对当前手机的了解，可以开发一个手机助手模块。通过这个模块，用户可以及时查看当前所用手机的基本信息。在本章的内容中，将详细介绍一个通用手机助手项目的基本实现流程，并详细阐述 Android 技术在实现过程的具体原理和注意事项。

15.1　项目分析

本项目实例的功能是，为用户提供显示当前手机信息的功能。在进行具体的编码工作之前，先对当前项目的构成模块和基本流程进行剖析，为后面的编码工作打好基础。

15.1.1　构成模块

具体来说，本项目实例主要包括如下 5 大模块信息。

1．系统模块信息

系统信息即整个系统的基本信息，主要包括如下 3 个方面。

- 操作系统的版本。
- 手机设备的系统信息。
- 运营商信息。

2．硬件模块信息

当前手机设备的硬件信息，主要包括如下 5 个方面。

- CPU 信息。
- 内存信息。
- 硬盘信息。
- 网络信息。
- 屏幕信息。

3．软件模块信息

当前手机设备内的各个基本软件的信息，通过本项目模块，会检索出当前手机设备内的各个基本软件状况。

4．运行时模块信息

即当前设备内正在运行的应用，主要包括如下 3 方面的应用信息。

- 正在运行的服务。
- 正在运行的任务。
- 正在运行的进程。

5. 文件模块信息

查看当前设备内各个文件的基本信息，包括了根目录和子目录下的信息。

上述模块的基本结构如图 15−1 所示。

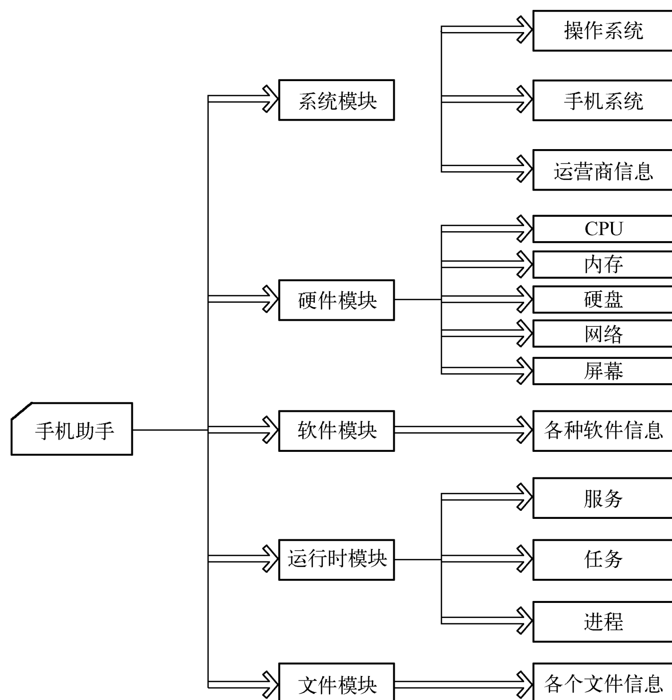


图 15−1 模块结构图

15.1.2 流程规划

本项目的具体实现流程如图 15−2 所示。

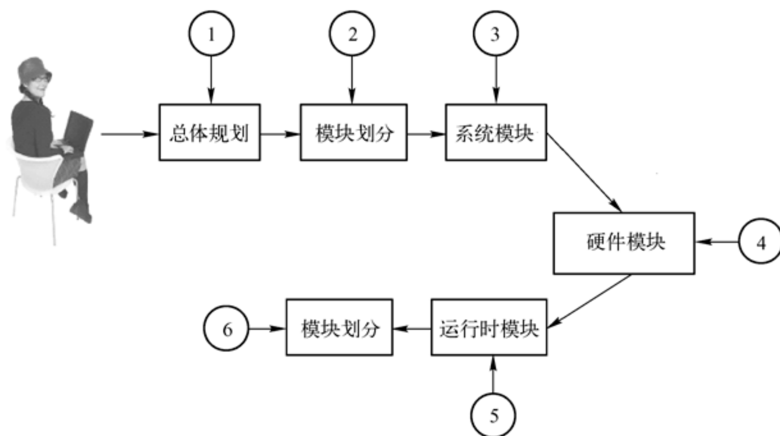


图 15−2 实现流程图



15.2　具体实现

从本节内容开始，将着重介绍本项目的具体实现过程。

15.2.1　系统主界面

系统主界面的布局文件为 main.xml，具体实现代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical" android:layout_width="fill_parent"
        android:layout_height="fill_parent">
        <ListView
              android:layout_width="fill_parent"
               android:layout_height="fill_parent"
               android:id="@+id/itemlist" />
</LinearLayout>
```

由此可以看出，系统主界面比较简单，只是通过 ListView 实现了分类信息的显示。

接下来看文件 infos.java，此文件是整个程序的入口，即程序的主界面。其具体实现流程如下。

1）通过 onCreate 方法调用 refreshListItems()来获取列表信息。

2）引用 buildListForSimpleAdapter()方法来填充相关数据，通过它可以来构建一个 List 列表，将需要的大分类名称和描述信息填充到里面。

3）通过 refreshListItems()方法引入 SimpleAdapter 适配方式，它使用了一个单独的布局文件 item_row.xml 作为模板文件。

4）重写单击每行的响应方法 onItemClick。

文件 infos.java 的具体代码如下所示。

```
package com.mobile.infos;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import com.mobile.infos.R;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
```

```
import android.widget.SimpleAdapter;
import android.widget.AdapterView.OnItemClickListener;

public class Infos extends Activity implements OnItemClickListener {
    private static final String TAG = "eoeInfosAssistant";
    ListView itemlist = null;
    List<Map<String, Object>> list;

    /** Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        itemlist = (ListView) findViewById(R.id.itemlist);
        refreshListItems();
    }

    private void refreshListItems() {
        list = buildListForSimpleAdapter();
        SimpleAdapter notes = new SimpleAdapter(this, list, R.layout.item_row,
                new String[] { "name", "desc", "img" }, new int[] { R.id.name,
                        R.id.desc, R.id.img });
        itemlist.setAdapter(notes);
        itemlist.setOnItemClickListener(this);
        itemlist.setSelection(0);
    }

    private List<Map<String, Object>> buildListForSimpleAdapter() {
        List<Map<String, Object>> list = new ArrayList<Map<String, Object>>(3);
        // Build a map for the attributes
        Map<String, Object> map = new HashMap<String, Object>();
        map.put("name", "系统信息");
        map.put("desc", "查看设备系统版本,运营商及其系统信息。");
        map.put("img", R.drawable.system);
        list.add(map);

        map = new HashMap<String, Object>();
        map.put("name", "硬件信息");
        map.put("desc", "查看包括 CPU,硬盘,内存等硬件信息。");
        map.put("img", R.drawable.hardware);
        list.add(map);

        map = new HashMap<String, Object>();
        map.put("name", "软件信息");
        map.put("desc", "查看已经安装的软件信息。");
        map.put("img", R.drawable.software);
```



```
            list.add(map);

            map = new HashMap<String, Object>();
            map.put("name", "运行时信息");
            map.put("desc", "查看设备运行时的信息。");
            map.put("img", R.drawable.running);
            list.add(map);

            map = new HashMap<String, Object>();
            map.put("name", "文件浏览器");
            map.put("desc", "浏览查看文件系统。");
            map.put("img", R.drawable.file_explorer);
            list.add(map);

            return list;
        }

        @Override
        public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
            Intent intent = new Intent();
            Log.i(TAG, "item clicked! [" + position + "]");
            switch (position) {
            case 0:
                intent.setClass(Infos.this, System.class);
                startActivity(intent);
                break;
            case 1:
                intent.setClass(Infos.this, Hardware.class);
                startActivity(intent);
                break;
            case 2:
                intent.setClass(Infos.this, Software.class);
                startActivity(intent);
                break;
            case 3:
                intent.setClass(Infos.this, Runing.class);
                startActivity(intent);
                break;
            case 4:
                intent.setClass(Infos.this, FSExplorer.class);
                startActivity(intent);
                break;
            }
        }
    }
```

布局文件 item_row.xml 的具体实现代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout
    xmlns:android="http://schemas.android.com/apk/res/android"
    android:id="@+id/vw1"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:padding="4px"
    android:orientation="horizontal">
    <ImageView android:id="@+id/img"
        android:layout_width="32px"
        android:layout_margin="4px"
        android:layout_height="32px"/>
   <LinearLayout
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:orientation="vertical">
        <TextView android:id="@+id/name"
            android:textSize="18sp"
            android:textStyle="bold"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"/>
        <TextView android:id="@+id/desc"
            android:textSize="14sp"
            android:layout_width="fill_parent"
            android:paddingLeft="20px"
            android:layout_height="wrap_content"/>
    </LinearLayout>
</LinearLayout>
```

通过上述布局文件，设置了主界面中每行的显示样式，即分别通过名字、示意图和简介来描述一行的内容。这样系统主界面就设计完毕了，执行后的效果如图 15-3 所示。



图 15-3 系统主界面

15.2.2 系统信息

主界面介绍完毕后，下面开始介绍各个子菜单的具体实现过程，本节将首先介绍系统信息模块的实现过程。当单击图 15-3 中的系统信息后，会来到系统信息界面，在此界面中会显示和系统有关的基本信息。主要包括如下 3 个方面的信息。

- ❑ 操作系统的版本。
- ❑ 手机设备的系统信息。
- ❑ 运营商信息。

第 1 步：先看布局文件 showinfo.xml，具体实现代码如下所示。



```
<?xml version="1.0" encoding="utf-8"?>
<ScrollView xmlns:android="http://schemas.android.com/apk/res/android"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content">
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
      android:orientation="vertical"
      android:layout_width="fill_parent"
      android:layout_height="fill_parent"
      android:padding="20px"
      >

      <TextView android:id="@+id/title"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:textSize="20sp"
            android:paddingBottom="8dip"
            android:text="" />

      <TextView android:id="@+id/info"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="" />

</LinearLayout>
</ScrollView>
```

通过上述代码，设置了具体的信息的显示样式。

第 2 步：再看处理文件 System.java，其实现流程如下。

1）填充信息，实现构造系统信息查看所需要的功能。

2）将需要查看的信息和相关参数及 Bundle 方式传递到下一个活动。

文件 System.java 的具体实现代码如下所示。

```
package com.mobile.infos;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import com.mobile.infos.R;
import com.mobile.infos.util.PreferencesUtil;



import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
```

```
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.AdapterView.OnItemClickListener;

public class System extends Activity implements OnItemClickListener{
	private static final String TAG = "System";

	ListView itemlist = null;
	List<Map<String, Object>> list;

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.infos);
		setTitle("系统信息");
		itemlist = (ListView) findViewById(R.id.itemlist);
		refreshListItems();
	}

	private void refreshListItems() {
		list = buildListForSimpleAdapter();
		SimpleAdapter notes = new SimpleAdapter(this, list, R.layout.info_row,
				new String[] { "name", "desc" }, new int[] { R.id.name,
						R.id.desc });
		itemlist.setAdapter(notes);
		itemlist.setOnItemClickListener(this);
		itemlist.setSelection(0);
	}

	private List<Map<String, Object>> buildListForSimpleAdapter() {
		List<Map<String, Object>> list = new ArrayList<Map<String, Object>>(3);
		// Build a map for the attributes
		Map<String, Object> map = new HashMap<String, Object>();

		map = new HashMap<String, Object>();
		map.put("id", PreferencesUtil.VER_INFO);
		map.put("name", "操作系统版本");
		map.put("desc", "读取/proc/version 信息");
		list.add(map);


		map = new HashMap<String, Object>();
		map.put("id", PreferencesUtil.SystemProperty);
```



```
                map.put("name", "系统信息");
                map.put("desc", "手机设备的系统信息。");
                // map.put("icon", R.drawable.mem);
                list.add(map);

                map = new HashMap<String, Object>();
                map.put("id", PreferencesUtil.TEL_STATUS);
                map.put("name", "运营商信息");
                map.put("desc", "手机网络的运营商信息。");
                list.add(map);

                return list;
        }

        @Override
        public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
                Intent intent = new Intent();
                Log.i(TAG, "item clicked! [" + position + "]");
                Bundle info = new Bundle();
                Map<String, Object> map = list.get(position);
                info.putInt("id",   (Integer) map.get("id"));
                info.putString("name", (String) map.get("name"));
                info.putString("desc", (String) map.get("desc"));
                info.putInt("position", position);
                intent.putExtra("android.intent.extra.info", info);
                intent.setClass(System.this, ShowInfo.class);
                startActivityForResult(intent, 0);
        }
}
```

这样，本模块的设计告一段落，执行后的将会显示系统的基本信息，如图 15-4 所示。



图 15-4　系统信息

第 3 步：用户可以继续单击图 15-4 中所显示的列表项，单击后将进一步显示出具体的信息，例如，单击“操作系统版本”后，将会显示当前移动设备的操作系统版本。上述功能是

通过 ShowInfo.java 实现的，其实现流程如下。

1）通过 revParams()来获取传递来的参数。

2）通过获取参数来调用某个方法来显示对应的要查看的数据信息。

文件 ShowInfo.java 的具体实现代码如下所示。

```
package com.mobile.infos;

import com.mobile.infos.R;
import com.mobile.infos.util.FetchData;
import com.mobile.infos.util.PreferencesUtil;

import android.app.Activity;
import android.app.ProgressDialog;
import android.content.Intent;
import android.os.Bundle;
import android.os.Handler;
import android.os.Message;
import android.util.Log;
import android.widget.TextView;

public class ShowInfo extends Activity implements Runnable {
	private static final String TAG = "ShowInfo";

	TextView info;
	TextView title;
	private ProgressDialog pd;
	public String info_datas;
	public boolean is_valid = false;
	public int _id = 0;
	public String _name = "";
	public int _position = 0;
	public int _ref = 0;

	@Override
	public void onCreate(Bundle savedInstanceState) {
		super.onCreate(savedInstanceState);
		setContentView(R.layout.showinfo);
		revParams();
		info = (TextView) findViewById(R.id.info);
		title = (TextView) findViewById(R.id.title);
		setTitle("eoeInfosAssistant: " + _name);
		title.setText(_name);
		load_data();
	}

	private void load_data() {
```



```
        pd = ProgressDialog.show(this, "Please Wait a moment..",
                "fetch info datas...", true, false);
        Thread thread = new Thread(this);
        thread.start();
    }

    // 接收传递进来的信息
    private void revParams() {
        Log.i(TAG, "revParams.");
        Intent startingIntent = getIntent();
        if (startingIntent != null) {
            Bundle infod = startingIntent
                    .getBundleExtra("android.intent.extra.info");
            if (infod == null) {
                is_valid = false;
            } else {
                _id = infod.getInt("id");
                _name = infod.getString("name");
                _position = infod.getInt("position");
                is_valid = true;
            }
        } else {
            is_valid = false;
        }
        Log.i(TAG, "_name:" + _name + ",_id="+_id);
    }

    @Override
    public void run() {
        if (_ref > 0) {
            try {
                Thread.sleep(1000);
            } catch (InterruptedException e) {
                Log.i("load_data", "e=" + e.toString());
            }
        }
        switch (_id) {
        case PreferencesUtil.CPU_INFO:
            info_datas = FetchData.fetch_cpu_info();
            break;
        case PreferencesUtil.DISK_INFO:
            info_datas = FetchData.fetch_disk_info();
            break;
        case PreferencesUtil.NET_STATUS:
            info_datas = FetchData.fetch_netstat_info();
            break;
```




```
            case PreferencesUtil.VER_INFO:
                info_datas = FetchData.fetch_version_info();
                break;
            case PreferencesUtil.DMESG_INFO:
                info_datas = FetchData.fetch_dmesg_info();
                break;
            case PreferencesUtil.RunningProcesses:
                info_datas = FetchData.fetch_process_info();
                break;
            case PreferencesUtil.NET_CONFIG:
                info_datas = FetchData.fetch_netcfg_info();
                break;
            case PreferencesUtil.MOUNT_INFO:
                info_datas = FetchData.fetch_mount_info();
                break;
            case PreferencesUtil.TEL_STATUS:
                info_datas = FetchData.fetch_tel_status(this);
                break;
            case PreferencesUtil.MEM_INFO:
                info_datas = FetchData.getMemoryInfo(this);
                break;
            case PreferencesUtil.SystemProperty:
                info_datas = FetchData.getSystemProperty();
                break;
            case PreferencesUtil.DisplayMetrics:
                info_datas = FetchData.getDisplayMetrics(this);
                break;
            case PreferencesUtil.RunningService:
                info_datas = FetchData.getRunningServicesInfo(this);
                break;
            case PreferencesUtil.RunningTasks:
                info_datas = FetchData.getRunningTasksInfo(this);
                break;
            }

            handler.sendEmptyMessage(0);
        }

        private Handler handler = new Handler() {
            public void handleMessage(Message msg) {
                pd.dismiss();
                info.setText(info_datas);
            }
        };
    }
```



第 4 步：可以根据需要来显示各种信息，下面分别介绍各个细分模块的具体实现。

（1）操作系统版本

操作系统版本基本信息保存在“/proc/version”中，读者可以直接调用，即通过 fetch_version_info()来调用。对应的调用代码如下所示。

```
public static String fetch_version_info() {
    String result = null;
    CMDExecute cmdexe = new CMDExecute();
    try {
        String[] args = { "/system/bin/cat", "/proc/version" };
        result = cmdexe.run(args, "/system/bin/");
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    return result;
}
```

这样就获得了操作系统版本信息，执行后效果如图 15-5 所示。



图 15-5　操作系统版本信息

（2）基本系统信息

Android 中可以通过方法 System.getProperty(propertyStr)来实现。对应的具体实现代码如下所示。

```
/**
 * 系统信息查看方法
 */
public static String getSystemProperty() {
    buffer = new StringBuffer();
    initProperty("java.vendor.url", "java.vendor.url");
    initProperty("java.class.path", "java.class.path");
    initProperty("user.home", "user.home");
    initProperty("java.class.version", "java.class.version");
    initProperty("os.version", "os.version");
    initProperty("java.vendor", "java.vendor");
    initProperty("user.dir", "user.dir");
    initProperty("user.timezone", "user.timezone");
```

```
            initProperty("path.separator", "path.separator");
            initProperty(" os.name", " os.name");
            initProperty("os.arch", "os.arch");
            initProperty("line.separator", "line.separator");
            initProperty("file.separator", "file.separator");
            initProperty("user.name", "user.name");
            initProperty("java.version", "java.version");
            initProperty("java.home", "java.home");
            return buffer.toString();
    }

    private static String initProperty(String description, String propertyStr) {
            if (buffer == null) {
                    buffer = new StringBuffer();
            }
            buffer.append(description).append(":");
            buffer.append(System.getProperty(propertyStr)).append("\n");
            return buffer.toString();
    }
```

这样，通过 System.getProperty(propertyStr)获取了基本的系统信息,执行后效果如图 15-6 所示。



图 15-6　基本系统信息

（3）运营商信息

运营商相关信息可以通过 Android 中的 TelephonyManger 来获取，对应的具体实现代码如下所示。

```
    public static String fetch_tel_status(Context cx) {
            String result = null;
```



```
        TelephonyManager tm = (TelephonyManager) cx
                .getSystemService(Context.TELEPHONY_SERVICE);//
        String str = "";
        str += "DeviceId(IMEI) = " + tm.getDeviceId() + "\n";
        str += "DeviceSoftwareVersion = " + tm.getDeviceSoftwareVersion()
                + "\n";
        str += "Line1Number = " + tm.getLine1Number() + "\n";
        str += "NetworkCountryIso = " + tm.getNetworkCountryIso() + "\n";
        str += "NetworkOperator = " + tm.getNetworkOperator() + "\n";
        str += "NetworkOperatorName = " + tm.getNetworkOperatorName() + "\n";
        str += "NetworkType = " + tm.getNetworkType() + "\n";
        str += "PhoneType = " + tm.getPhoneType() + "\n";
        str += "SimCountryIso = " + tm.getSimCountryIso() + "\n";
        str += "SimOperator = " + tm.getSimOperator() + "\n";
        str += "SimOperatorName = " + tm.getSimOperatorName() + "\n";
        str += "SimSerialNumber = " + tm.getSimSerialNumber() + "\n";
        str += "SimState = " + tm.getSimState() + "\n";
        str += "SubscriberId(IMSI) = " + tm.getSubscriberId() + "\n";
        str += "VoiceMailNumber = " + tm.getVoiceMailNumber() + "\n";

        int mcc = cx.getResources().getConfiguration().mcc;
        int mnc = cx.getResources().getConfiguration().mnc;
        str += "IMSI MCC (Mobile Country Code):" + String.valueOf(mcc) + "\n";
        str += "IMSI MNC (Mobile Network Code):" + String.valueOf(mnc) + "\n";
        result = str;
        return result;
    }
```

这样，通过 getSystemService 获取了 TelephonyManger 对象，从而获取了运营商的基本信息。执行后效果如图 15-7 所示。



图 15-7　运营商信息

15.2.3 硬件信息

本节将介绍硬件信息模块的实现过程。当单击图 15-3 中的“硬件信息”后，会来到硬件信息界面，如图 15-8 所示。



图 15-8　硬件信息

图 15-8 界面中会显示和硬件有关的基本信息，主要包括如下 5 个方面的信息。

- CPU 信息。
- 内存信息。
- 硬盘信息。
- 网络信息。
- 屏幕信息。

1．获取CPU信息

Android 中可以在“/proc/cpuinfo”中获取 CPU 的信息，即通过调用 CMDExecute 执行系统的 cat 命令，读取“/proc/cpuinfo”的内容，这样显示的就是 CPU 的信息。对应的具体代码如下所示。

```
public static String fetch_cpu_info() {
    String result = null;
    CMDExecute cmdexe = new CMDExecute();
    try {
        String[] args = { "/system/bin/cat", "/proc/cpuinfo" };
        result = cmdexe.run(args, "/system/bin/");
        Log.i("result", "result=" + result);
    } catch (IOException ex) {
        ex.printStackTrace();
    }
    return result;
}
```

这样就获取了 CPU 的信息，执行后的效果如图 15-9 所示。





图 15-9　获取的 CPU 信息

2．获取内存信息

Android 中可以在"/proc/cpuinfo"中获取 CPU 的信息，也可以通过 getMemoryInfo(Context context)来获取 ActivityManager.MemoryInfo 对象的方法来获取。对应的具体代码如下所示。

```
/**
 * 系统内存情况查看
 */
public static String getMemoryInfo(Context context) {
        StringBuffer memoryInfo = new StringBuffer();
        final ActivityManager activityManager = (ActivityManager) context
                        .getSystemService(Context.ACTIVITY_SERVICE);

        ActivityManager.MemoryInfo outInfo = new ActivityManager.MemoryInfo();
        activityManager.getMemoryInfo(outInfo);
        memoryInfo.append("\nTotal Available Memory :").append(
                        outInfo.availMem >> 10).append("k");
        memoryInfo.append("\nTotal Available Memory :").append(
                        outInfo.availMem >> 20).append("M");
        memoryInfo.append("\nIn low memory situation:").append(
                        outInfo.lowMemory);

        String result = null;
        CMDExecute cmdexe = new CMDExecute();
        try {
                String[] args = { "/system/bin/cat","/proc/meminfo" };
                result = cmdexe.run(args, "/system/bin/");
        } catch (IOException ex) {
                Log.i("fetch_process_info", "ex=" + ex.toString());
        }

        return memoryInfo.toString()+"\n\n"+result;
}
```

在上述代码中，首先通过 getMemoryInfo(Context context)获取了可用的内存，然后通过查

看“/proc/cpuinfo”的内容来获取更加详细的信息。执行后的效果如图 15-10 所示。

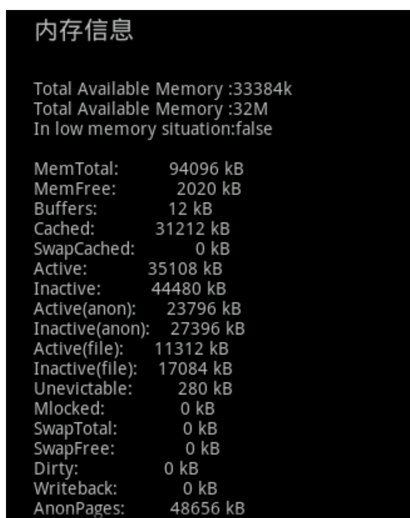


图 15-10　获取的内存信息

3．获取硬盘信息

Android 领域中通常通过 df 命令获取硬盘信息，用户需要直接执行命令，然后获取命令的返回就完成了。具体实现流程和前面模块的类似，对应的具体代码如下所示。

```
public static String fetch_disk_info() {
        String result = null;
        CMDExecute cmdexe = new CMDExecute();
        try {
                String[] args = { "/system/bin/df" };
                result = cmdexe.run(args, "/system/bin/");
        } catch (IOException ex) {
                ex.printStackTrace();
        }
        return result;
}
```

在上述代码中，通过 CMDExecute 调用了系统内的“/system/bin/df”命令来获取了其磁盘信息。执行后的效果如图 15-11 所示。

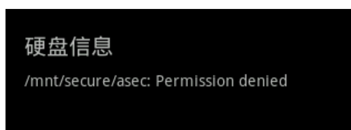


图 15-11　获取硬盘信息

4．获取网络信息

Android 中通过读取“/system/bin/netcfg”的信息来获取网络信息，对应的具体代码如下所示。

```
public static String fetch_netstat_info() {
        String result = null;
        CMDExecute cmdexe = new CMDExecute();
        try {
```



```
            String[] args = { "/system/bin/netstat" };
            result = cmdexe.run(args, "/system/bin/");
        } catch (IOException ex) {
            ex.printStackTrace();
        }
        return result;
    }
```

在上述代码中，通过“/system/bin/netcfg”命令来获取了网络信息。

5．获取屏幕信息

Android 中有一个类 DisplayMetrics，可以分别通过 getApplicationContext()、getResources()、getDisplayMetrics()初始化处理，进而读取其屏幕的信息（如宽度、高度），对应的具体代码如下所示。

```
    public static String getDisplayMetrics(Context cx) {
        String str = "";
        DisplayMetrics dm = new DisplayMetrics();
        dm = cx.getApplicationContext().getResources().getDisplayMetrics();
        int screenWidth = dm.widthPixels;
        int screenHeight = dm.heightPixels;
        float density = dm.density;
        float xdpi = dm.xdpi;
        float ydpi = dm.ydpi;
        str += "The absolute width:" + String.valueOf(screenWidth) + "pixels\n";
        str += "The absolute heightin:" + String.valueOf(screenHeight)
                + "pixels\n";
        str += "The logical density of the display.:" + String.valueOf(density)
                + "\n";
        str += "X dimension :" + String.valueOf(xdpi) + "pixels per inch\n";
        str += "Y dimension :" + String.valueOf(ydpi) + "pixels per inch\n";
        return str;
    }
```

执行后的效果如图 15-12 所示。



图 15-12　获取屏幕信息

15.2.4　软件信息

本节将介绍软件信息模块的实现过程。当单击图 15-13 中的“软件信息”后，会来到软件信息界面，如图 15-13 所示。



图 15-13　软件信息

图 15-13 界面中会显示本设备内安装的软件信息。上述功能是由文件 Software.java 实现的，具体实现流程如下。

1）使用 ProgressDialog 实现一个进度条，并设置了提示信息。对应代码如下。

```
public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
        setContentView(R.layout.infos);
        setTitle("软件信息");
        itemlist = (ListView) findViewById(R.id.itemlist);
        pd = ProgressDialog.show(this, "请稍候..", "正在收集你已经安装的软件信息...", true,
                false);
        Thread thread = new Thread(this);
        thread.start();
 }
```

2）线程方法 run()实现其执行的逻辑。对应代码如下。

```
@Override
public void run() {
        fetch_installed_apps();
      handler.sendEmptyMessage(0);
}
```

3）调用自定义的 fetch_installed_apps()方法获取已安装的应用信息。对应代码如下。

```
public List fetch_installed_apps(){
        List<ApplicationInfo> packages = getPackageManager().getInstalledApplications(0);
```



```
            list = new ArrayList<Map<String, Object>>(
                        packages.size());
            Iterator<ApplicationInfo> l = packages.iterator();

            while (l.hasNext()) {
                  Map<String, Object> map = new HashMap<String, Object>();
                  ApplicationInfo app = (ApplicationInfo) l.next();
                  String packageName = app.packageName;
                 String label = "";
                 try {
                       label = getPackageManager().getApplicationLabel(app).toString();
                 } catch (Exception e) {
                  Log.i("Exception",e.toString());
                 }
                  map = new HashMap<String, Object>();
                  map.put("name", label);
                  map.put("desc", packageName);
                  list.add(map);
            }
            return list;
      }
```

在上述代码中，通过使用 getPackageManager().getInstalledApplications(0)获取了已经安装的软件信息，进而构造用来显示的列表(list)对象，同时界面通过进度条显示提示信息。

4）调用 handle.sendEmptyMessage(0)语句给 handle 发送一个通知信息，handle 的实现代码如下。

```
      private Handler handler = new Handler() {
            public void handleMessage(Message msg) {
                  refreshListItems();
                  pd.dismiss();
            }
      };
```

5）接收到 run()后先调用 refreshListItems()方法来显示列表，然后调用进度条 ProgressDialog 的 dismiss 方法使其等待提示的消失。refreshListItems()方法的实现代码如下。

```
     private void refreshListItems() {
            list = fetch_installed_apps();
            SimpleAdapter notes = new SimpleAdapter(this, list, R.layout.info_row,
                        new String[] { "name", "desc" }, new int[] { R.id.name,
                                    R.id.desc });
            itemlist.setAdapter(notes);
            setTitle("软件信息,已经安装"+list.size()+"款应用。");
     }
```

通过上面的代码编写工作，实现了软件信息模块。在运行后会首先显示一个进度条界面，

如图 15-14 所示，然后才会显示图 15-13 的软件信息界面。



图 15-14　进度条

15.2.5　运行时信息

本节将介绍运行时信息模块的实现过程。当单击图 15-3 中的“运行时信息”后，会显示当前正在运行的信息界面，如图 15-15 所示。



图 15-15　正在运行

由图 15-15 界面所示，系统能够获取如下 3 个方面的信息。

- 正在运行的服务。
- 正在运行的任务。
- 正在运行的进程。

先看布局文件 runing.xml，其具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <ListView
```



```
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            android:id="@+id/itemlist" />
</LinearLayout>
```

处理文件 Runing.java 的具体代码如下所示。

```
package com.mobile.infos;

import java.util.ArrayList;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import com.mobile.infos.R;
import com.mobile.infos.util.PreferencesUtil;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.AdapterView;
import android.widget.ListView;
import android.widget.SimpleAdapter;
import android.widget.AdapterView.OnItemClickListener;

public class Runing extends Activity implements OnItemClickListener {
    private static final String TAG = "System";

    ListView itemlist = null;
    List<Map<String, Object>> list;

    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.runing);
        setTitle("运行时信息");
        itemlist = (ListView) findViewById(R.id.itemlist);
        refreshListItems();
    }

    private void refreshListItems() {
        list = buildListForSimpleAdapter();
        SimpleAdapter notes = new SimpleAdapter(this, list, R.layout.info_row,
                new String[] { "name", "desc" }, new int[] { R.id.name,
                        R.id.desc });
```

```
		itemlist.setAdapter(notes);
		itemlist.setOnItemClickListener(this);
		itemlist.setSelection(0);
	}

	private List<Map<String, Object>> buildListForSimpleAdapter() {
		List<Map<String, Object>> list = new ArrayList<Map<String, Object>>(3);
		// Build a map for the attributes
		Map<String, Object> map = new HashMap<String, Object>();

		map = new HashMap<String, Object>();
		map.put("id", PreferencesUtil.RunningService);
		map.put("name", "运行的 service");
		map.put("desc", "获取正在运行的后台服务。");
		list.add(map);


		map = new HashMap<String, Object>();
		map.put("id", PreferencesUtil.RunningTasks);
		map.put("name", "运行的 Task");
		map.put("desc", "获取正在运行的任务。");
		list.add(map);

		map = new HashMap<String, Object>();
		map.put("id", PreferencesUtil.RunningProcesses);
		map.put("name", "进程信息");
		map.put("desc", "获取正在运行的进程。");
		list.add(map);


		return list;
	}

	@Override
	public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
		Intent intent = new Intent();
		Log.i(TAG, "item clicked! [" + position + "]");
		Bundle info = new Bundle();
		Map<String, Object> map = list.get(position);
		info.putInt("id",    (Integer) map.get("id"));
		info.putString("name", (String) map.get("name"));
		info.putString("desc", (String) map.get("desc"));
		info.putInt("position", position);
		intent.putExtra("android.intent.extra.info", info);
		intent.setClass(Runing.this, ShowInfo.class);
		startActivityForResult(intent, 0);
```



```
            }
        }
```

下面开始介绍各子菜单模块的具体实现流程。

1．正在运行的服务

通过调用 context.getSystemService(Context.ACTIVITY_SERVICE)来获取 ActivityManage，进而通过系统提供的方法 getRunningServices（int maxNum）获取正在运行的服务列表<RunningServiceInfo>，然后对其结果进行进一步分析，这样就得到了对应的进程名和其他信息。对应的具体代码如下所示。

```
        public static String getRunningServicesInfo(Context context) {
            StringBuffer serviceInfo = new StringBuffer();
            final ActivityManager activityManager = (ActivityManager) context
                        .getSystemService(Context.ACTIVITY_SERVICE);
            List<RunningServiceInfo> services = activityManager.getRunningServices(100);

            Iterator<RunningServiceInfo> l = services.iterator();
            while (l.hasNext()) {
                RunningServiceInfo si = (RunningServiceInfo) l.next();
                serviceInfo.append("pid: ").append(si.pid);
                serviceInfo.append("\nprocess: ").append(si.process);
                serviceInfo.append("\nservice: ").append(si.service);
                serviceInfo.append("\ncrashCount: ").append(si.crashCount);
                serviceInfo.append("\nclientCount: ").append(si.clientCount);
                serviceInfo.append("\nactiveSince: ").
append(ToolHelper.formatData(si.activeSince));
                serviceInfo.append("\nlastActivityTime:
").append(ToolHelper.formatData(si.lastActivityTime));
                serviceInfo.append("\n\n");
            }
            return serviceInfo.toString();
        }
```

这样就获取了正在运行的服务信息。

2．正在运行的Task

获取正在运行的 Task 和前面的实现类似，其调用的是 activityManager.getRunningTasks(int maxNum)来获取正在运行的任务信息列表，从而进一步分析、显示其任务信息。对应的具体代码如下所示。

```
        public static String getRunningTasksInfo(Context context) {
            StringBuffer sInfo = new StringBuffer();
            final ActivityManager activityManager = (ActivityManager) context
                        .getSystemService(Context.ACTIVITY_SERVICE);
            List<RunningTaskInfo> tasks = activityManager.getRunningTasks(100);
            Iterator<RunningTaskInfo> l = tasks.iterator();
            while (l.hasNext()) {
```

```
                RunningTaskInfo ti = (RunningTaskInfo) l.next();
                sInfo.append("id: ").append(ti.id);
                sInfo.append("\nbaseActivity: ").append(ti.baseActivity.flattenToString());
                sInfo.append("\nnumActivities: ").append(ti.numActivities);
                sInfo.append("\nnumRunning: ").append(ti.numRunning);
                sInfo.append("\ndescription: ").append(ti.description);
                sInfo.append("\n\n");
            }
            return sInfo.toString();
        }
```

这样，通过 activityManager.getRunningTasks(100)获取了当前设备中的 Task 列表，如图 15-16 所示。

运行任务

id: 3
baseActivity: com.secretary/com.secretary.Infos
numActivities: 3
numRunning: 3
description: null

id: 2
baseActivity: com.android.launcher/com.android.launcher2.Launcher
numActivities: 1
numRunning: 1
description: null

图 15-16　正在运行的 Task

3. 正在运行的进程

获取正在运行的进程的实现方法比较简单，对应的具体代码如下所示。

```
    public static String fetch_process_info() {
        Log.i("fetch_process_info", "start....");
        String result = null;
        CMDExecute cmdexe = new CMDExecute();
        try {
            String[] args = { "/system/bin/top", "-n", "1" };
            result = cmdexe.run(args, "/system/bin/");
        } catch (IOException ex) {
            Log.i("fetch_process_info", "ex=" + ex.toString());
        }
        return result;
    }
```

执行后的效果如图 15-17 所示。



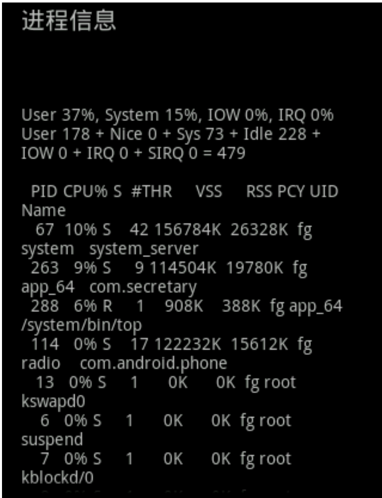


图 15-17　正在运行的进程

15.2.6　文件浏览器信息

本节将介绍文件浏览器模块的实现过程。当单击图 15-3 中的“文件浏览器”后，会显示当前系统内所有的和文件有关的信息，如图 15-18 所示。

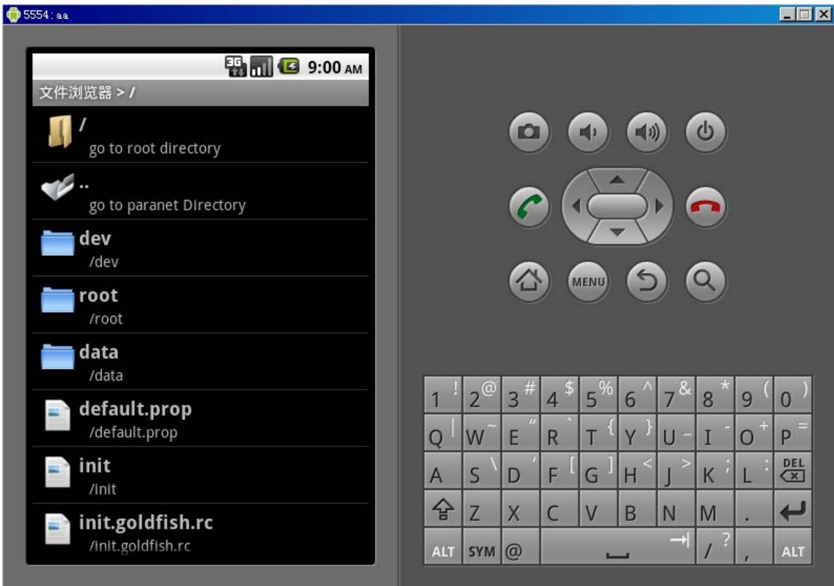


图 15-18　文件信息

先看布局文件 files.xml，其具体代码代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical" android:layout_width="fill_parent"
```

```
        android:layout_height="fill_parent">
        <ListView
            android:layout_width="fill_parent"
             android:layout_height="fill_parent"
             android:id="@+id/itemlist" />
</LinearLayout>
```

再看处理文件 wenjian.java，其实现流程如下。

1）编写 onCreate 方法，代码如下。

```
public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.files);
        setTitle("文件浏览器");
        itemlist = (ListView) findViewById(R.id.itemlist);
        refreshListItems(path);
}
```

2）调用 refreshListItems(String path)方法来获取文件列表，具体代码如下。

```
private void refreshListItems(String path) {
        setTitle("文件浏览器 > "+path);
        list = buildListForSimpleAdapter(path);
        SimpleAdapter notes = new SimpleAdapter(this, list, R.layout.file_row,
                        new String[] { "name", "path" ,"img"}, new int[] { R.id.name,
                                        R.id.desc ,R.id.img});
        itemlist.setAdapter(notes);
        itemlist.setOnItemClickListener(this);
        itemlist.setSelection(0);
}
```

3）调用 buildListForSimpleAdapter(String path)方法来获取文件列表，具体代码如下。

```
private List<Map<String, Object>> buildListForSimpleAdapter(String path) {
        File[] files = new File(path).listFiles();
        List<Map<String, Object>> list = new ArrayList<Map<String, Object>>(files.length);
        Map<String, Object> root = new HashMap<String, Object>();
        root.put("name", "/");
        root.put("img", R.drawable.file_root);
        root.put("path", "go to root directory");
        list.add(root);
        Map<String, Object> pmap = new HashMap<String, Object>();
        pmap.put("name", "..");
        pmap.put("img", R.drawable.file_paranet);
        pmap.put("path", "go to paranet Directory");
        list.add(pmap);
        for (File file : files){
```



```
                Map<String, Object> map = new HashMap<String, Object>();
                if(file.isDirectory()){
                    map.put("img", R.drawable.directory);
                }else{
                    map.put("img", R.drawable.file_doc);
                }
                map.put("name", file.getName());
                map.put("path", file.getPath());
                list.add(map);
            }
            return list;
        }
```

在上述代码中，使用 File(path).listFiles()方法来获取了文件和文件夹的列表，最后存入 List 显示出来即可。

4）编写单击响应的处理事件代码，这样将进入到子目录，并获取和显示其内部文件夹及信息，具体代码如下。

```
        @Override
        public void onItemClick(AdapterView<?> parent, View v, int position, long id) {
            Log.i(TAG, "item clicked! [" + position + "]");
            if (position == 0) {
                path = "/";
                refreshListItems(path);
            }else if(position == 1){
                goToParent();
            } else {
                path = (String) list.get(position).get("path");
                File file = new File(path);
                if (file.isDirectory())
                    refreshListItems(path);
                else
                    Toast.makeText(wenjian.this,
                            getString(R.string.is_file),
                            Toast.LENGTH_SHORT).show();
            }

        }
```

在上述代码中，如果单击“/”行，则使用 refreshListItems(path)来获取根目录下的文件系统；如果单击“上一级”行，则调用 goToParent()来显示其上一级的文件系统目录。goToParent()方法的具体实现代码如下。

```
        private void goToParent() {
            File file = new File(path);
            File str_pa = file.getParentFile();
```

```
            if(str_pa == null){
                Toast.makeText(wenjian.this,
                        getString(R.string.is_root_dir),
                        Toast.LENGTH_SHORT).show();
                refreshListItems(path);
            }else{
                path = str_pa.getAbsolutePath();
                refreshListItems(path);
            }
        }
```

15.2.7 AndroidManifest.xml获取权限

此处的文件 AndroidManifest.xml，还有一个很重要的功能，即申请电话状态的权限，在里面添加权限需求。具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<manifest xmlns:android="http://schemas.android.com/apk/res/android"
      package="com.mobile.infos"
      android:versionCode="2"
      android:versionName="1.1.0">
    <application android:icon="@drawable/icon" android:label="@string/app_name">
        <activity android:name=".Infos"
                  android:label="@string/app_name">
            <intent-filter>
                <action android:name="android.intent.action.MAIN" />
                <category android:name="android.intent.category.LAUNCHER" />
            </intent-filter>
        </activity>
        <activity android:name="System"></activity>
        <activity android:name="Hardware"></activity>
        <activity android:name="Software"></activity>
        <activity android:name="Runing"></activity>
        <activity android:name="ShowInfo"></activity>
         <activity android:name="wenjian"></activity>

    </application>
<uses-permission android:name="android.permission.READ_PHONE_STATE"></uses-permission>
<uses-permission android:name="android.permission.GET_TASKS"></uses-permission>
</manifest>
```

这样，在上述代码中就包含了所有的 Activity 和需要申请的权限。

第 16 章　Map地图综合应用

Android 提供的地图（Map）功能可能是广大开发者非常关心的一个部分。到目前为止，开发内嵌式地图应用的软件是相当的困难，而且往往还需要支付很高的地图厂商的版权费用，加之手机上 GPS 功能的不完善，导致很多可以基于当前位置来开发功能的软件少之又少。在本章的内容中，将详细介绍 Map 地图在 Android 中的应用，并通过一个综合实例的实现过程来讲解具体实现流程。

16.1　项目分析

本项目实例的功能是，为用户提供需要的目标定位处理，即用户设置一个目标后，可以在后台启动一个 Service，能够定时读取 GPS 数据以获得用户目前所在的位置信息，并将其保存在数据库中。用户也可以选择其他目标信息，也能够将这些轨迹显示在 Map 地图上。本项目的具体实现过程如图 16-1 所示。

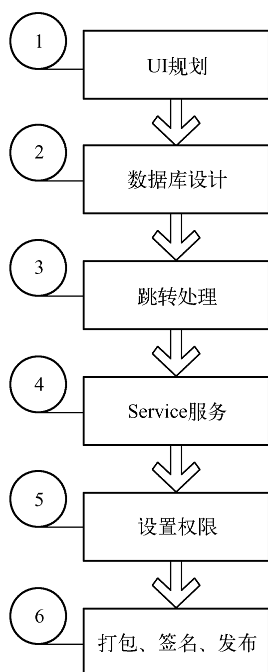


图 16-1　实现流程

16.1.1 规划UI界面

根据系统需求，UI 界面结构如图 16-2 所示。

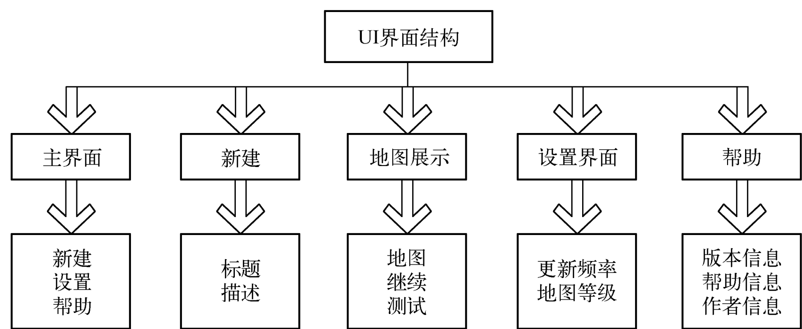


图 16-2 UI 界面结构

16.1.2 数据存储设计

数据存储既可以通过文件系统实现，也可以通过专用数据库工具实现。但是为了保证系统的日后维护工作，本项目使用数据库工具方式。本项目使用的是最常见的 SQlite。

根据前面介绍的系统需求分析，本系统用到了 3 类数据，一种是目标名，一种是每次追踪时的位置信息，另外一种是配置信息。为此，在本系统需要设计 3 个表来存储数据，具体说明如表 16-1、表 16-2 所示。

表 16-1 Tracks 用于存储目标信息

属 性	类 型	说 明
id	INTEGER	主键
name	TEXT	名
desc	TEXT	说明
distance	LONG	距离
tracked_time	LONG	时间
locates_count	INTEGER	点数
created_at	INTEGER	创建时间
update_at	INTEGER	更新时间
avg_speed	LONG	平均速度
max_speed	LONG	最大速度

表 16-2 Locats 用于存储目标的位置信息

属 性	类 型	说 明
id	INTEGER	主键
track_id	INTEGER	跟踪的目标 ID
longitude	TEXT	维度
latiude	TEXT	经度
altitude	TEXT	偏差
created_at	INTEGER	创建时间



16.2　具体实现

从本节内容开始，将着重介绍本项目的具体实现过程。

16.2.1　新建工程

打开 Eclipse，依次单击“File”→“New”→“Android Project”，新建一个名为“map”的工程文件，如图 16-3 所示。

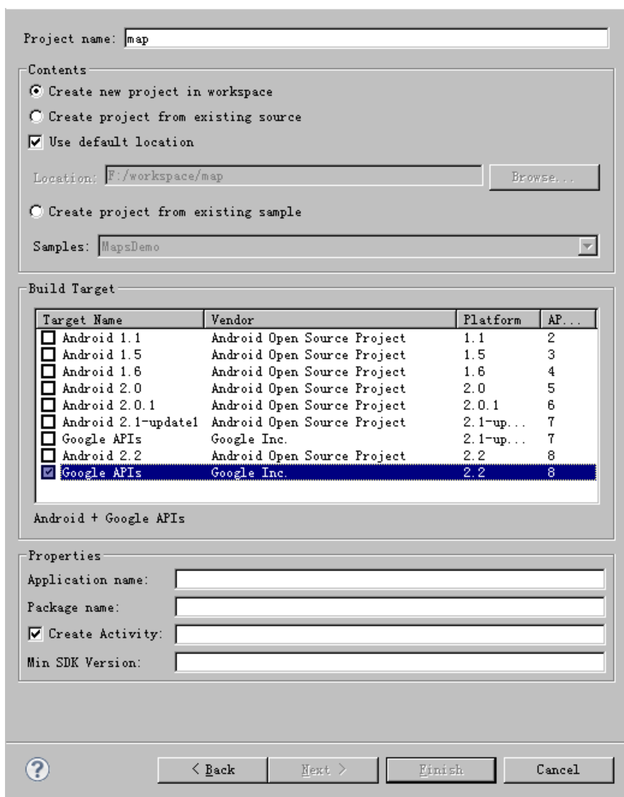


图 16-3　新建工程

16.2.2　主界面

主界面即项目执行后首先显示的界面，具体流程如下。

1）编写主布局文件 main.xml，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<TextView
```

```
        android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/title"
        />

        <ListView android:id="@id/android:list"
              android:layout_width="fill_parent"
               android:layout_height="wrap_content"
               android:drawSelectorOnTop="false" />

        <TextView android:id="@+id/android:empty"
               android:layout_width="wrap_content"
               android:layout_height="wrap_content"
               android:text="@string/start" />
</LinearLayout>
```

2）编写一个“历史记录”的列表信息，只显示系统数据库内的前 10 条数据。具体功能是在文件 string.xml 实现的，具体代码如下所示。

```
<string name="title">历史记录:</string>
<string name="app_name">aaa</string>
<string name="app_title">bbbb</string>
<string name="menu_main">主页</string>
<string name="menu_new">新建</string>
<string name="menu_con">继续</string>
<string name="menu_del">删除</string>
<string name="menu_setting">设置</string>
<string name="menu_helps">帮助</string>
<string name="menu_back">返回</string>
<string name="menu_exit">退出</string>
```

3）编写 onCreate 方法，具体代码如下。

```
@Override
public void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.main);
    render_tracks();
}
```

在上述代码中，需要将以往的历史记录从数据库中读取出来，显示在列表中去，然后使用 render_tracks()方法将数据库的历史记录读取出来，并更新到列表中去。

4）在 iTracks.java 编写实现菜单的代码，主要代码如下所示。

```
//定义菜单需要的常量
private static final int MENU_NEW = Menu.FIRST + 1;
private static final int MENU_CON = MENU_NEW + 1;
```



```
private static final int MENU_SETTING = MENU_CON + 1;
private static final int MENU_HELPS = MENU_SETTING + 1;
private static final int MENU_EXIT = MENU_HELPS + 1;
// 初始化菜单
@Override
public boolean onCreateOptionsMenu(Menu menu) {
    Log.d(TAG, "onCreateOptionsMenu");

    super.onCreateOptionsMenu(menu);

    menu.add(0, MENU_NEW, 0, R.string.menu_new).setIcon(
            R.drawable.new_track).setAlphabeticShortcut('N');
    menu.add(0, MENU_CON, 0, R.string.menu_con).setIcon(
            R.drawable.con_track).setAlphabeticShortcut('C');
    menu.add(0, MENU_SETTING, 0, R.string.menu_setting).setIcon(
            R.drawable.setting).setAlphabeticShortcut('S');
    menu.add(0, MENU_HELPS, 0, R.string.menu_helps).setIcon(
            R.drawable.helps).setAlphabeticShortcut('H');
    menu.add(0, MENU_EXIT, 0, R.string.menu_exit).setIcon(
            R.drawable.exit).setAlphabeticShortcut('E');
    return true;
}

// 当一个菜单被选中的时候调用
@Override
public boolean onOptionsItemSelected(MenuItem item) {
    Intent intent = new Intent();
    switch (item.getItemId()) {
    case MENU_NEW:
        intent.setClass(iTracks.this, NewTrack.class);
        startActivity(intent);
        return true;
    case MENU_CON:
        //TODO: 继续跟踪选择的记录
        conTrackService();
        return true;
    case MENU_SETTING:
        intent.setClass(iTracks.this, Setting.class);
        startActivity(intent);
        return true;
    case MENU_HELPS:
        intent.setClass(iTracks.this, Helps.class);
        startActivity(intent);
        return true;
    case MENU_EXIT:
        finish();
```

```
                break;
            }
            return true;
        }
```

通过上述代码，创建了菜单框架和菜单被选中后的响应方法。至此，主界面的主要代码介绍完毕，执行后的效果如图 16-4 所示。



图 16-4　主界面

16.2.3　新建界面

在图 16-4 中单击“新建”按钮后，会弹出新建目标记录界面。本模块的具体实现流程如下。

1）编写布局文件 new_track.xml，主要代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
    android:orientation="vertical" android:layout_width="fill_parent"
    android:layout_height="fill_parent">
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/new_tips" />
    <TextView android:layout_width="fill_parent"
        android:layout_height="wrap_content"
        android:text="@string/new_name" />
    <EditText android:id="@+id/new_name"
    android:layout_width="fill_parent"
```



```
            android:layout_height="wrap_content"
            android:text="" />
        <TextView android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:text="@string/new_desc" />
        <EditText android:id="@+id/new_desc"
            android:layout_width="fill_parent"
            android:layout_height="wrap_content"
            android:layout_weight="1"
            android:scrollbars="vertical"/>
        <Button android:id="@+id/new_submit"
            android:layout_width="wrap_content"
            android:layout_height="wrap_content"
            android:text="@string/new_submit" />
</LinearLayout>
```

在上述代码中，分别用 TextView 来显示提示信息，用 EditText 来接收用户的输入。

2）编写处理文件 NewTrack.java，具体代码如下所示。

```
package com.iceskysl.map;


import com.iceskysl.map.R;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.util.Log;
import android.view.View;
import android.widget.Button;
import android.widget.EditText;
import android.widget.Toast;

public class NewTrack extends Activity {
      private static final String TAG = "NewTrack";
    private Button button_new;
    private EditText field_new_name;
    private EditText field_new_desc;

    private TrackDbAdapter mDbHelper;

    /** Called when the activity is first created. */
      @Override
      public void onCreate(Bundle savedInstanceState) {
            super.onCreate(savedInstanceState);
            setContentView(R.layout.new_track);
          setTitle(R.string.menu_new);
```

```
            findViews();
            setListensers();

            mDbHelper = new TrackDbAdapter(this);
            mDbHelper.open();
        }

        @Override
        protected void onStop(){
            super.onStop();
            Log.d(TAG, "onStop");
            mDbHelper.close();
        }

        private void findViews() {
            Log.d(TAG, "find Views");
            field_new_name = (EditText) findViewById(R.id.new_name);
            field_new_desc = (EditText) findViewById(R.id.new_desc);
            button_new = (Button) findViewById(R.id.new_submit);
        }

        // Listen for button clicks
        private void setListensers() {
            Log.d(TAG, "set Listensers");
            button_new.setOnClickListener(new_track);
        }

        private Button.OnClickListener new_track = new Button.OnClickListener() {
            public void onClick(View v) {
                Log.d(TAG, "onClick new_track..");
                try {
                    String name = (field_new_name.getText().toString());
                    String desc = (field_new_desc.getText()
                            .toString());
                    if (name.equals("")) {
                        Toast.makeText(NewTrack.this,
                                getString(R.string.new_name_null),
                                Toast.LENGTH_SHORT).show();
                    } else {
                        // TODO  调用存储接口保存到数据库并启动 service
                        Long row_id = mDbHelper.createTrack(name, desc);
                        Log.d(TAG, "row_id="+row_id);

                        Intent intent = new Intent();
                        intent.setClass(NewTrack.this, ShowTrack.class);
                        intent.putExtra(TrackDbAdapter.KEY_ROWID, row_id);
```



```
                        intent.putExtra(TrackDbAdapter.NAME, name);
                        intent.putExtra(TrackDbAdapter.DESC, desc);

                        startActivity(intent);
                    }
                } catch (Exception err) {
                    Log.e(TAG, "error: " + err.toString());
                    Toast.makeText(NewTrack.this, getString(R.string.new_fail),
                            Toast.LENGTH_SHORT).show();
                }
            }
        };
    }
```

在上述代码中，首先在方法 onCreate 中设置了其关联的 layout；然后调用 findViewsByid()来获取名字和 EditText 组件，并获取提交按钮；最后，定义了一个 Button.OnClickListener new_track 对象，实现其 onClick 方法。

至此，本模块的主要功能设计完毕，执行后的效果如图 16-5 所示。

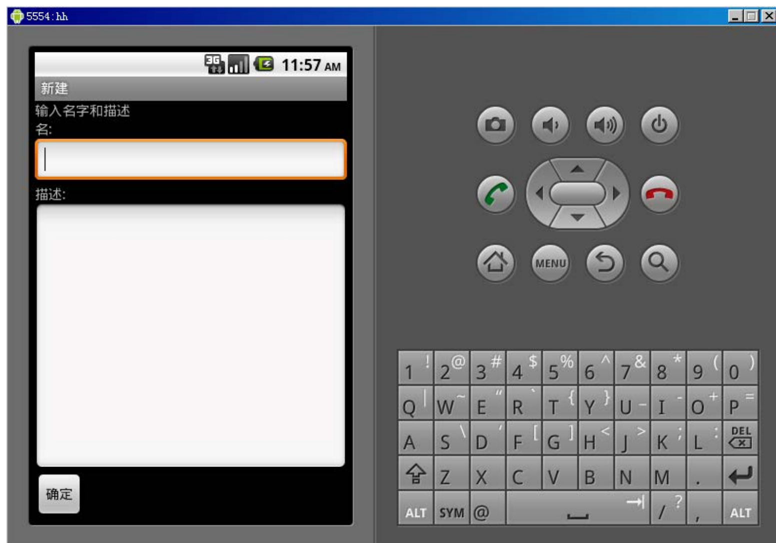


图 16-5　新建界面

16.2.4　设置界面

当在图 16-4 中单击“设置”按钮后，会弹出系统设置界面。本模块的具体实现流程如下。

1）编写布局文件 setting.xml，主要代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
        android:orientation="vertical" android:layout_width="fill_parent"
        android:layout_height="fill_parent">
```

```
            <TextView android:id="@+id/setting_tips"
                 android:layout_width="fill_parent"
                  android:layout_height="wrap_content"
                  android:text="" />
            <TextView android:layout_width="fill_parent"
                  android:layout_height="wrap_content"
                  android:text="@string/setting_gps" />
                       <Spinner android:id="@+id/setting_gps"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:drawSelectorOnTop="true"
                android:prompt="@string/spinner_gps_prompt"
           />
            <TextView android:layout_width="fill_parent"
                  android:layout_height="wrap_content"
                  android:text="@string/setting_map_level" />
           <Spinner android:id="@+id/setting_map_level"
                android:layout_width="fill_parent"
                android:layout_height="wrap_content"
                android:drawSelectorOnTop="true"
                android:prompt="@string/spinner_map_prompt"
           />
            <Button android:id="@+id/setting_submit"
                  android:layout_width="wrap_content"
                  android:layout_height="wrap_content"
                  android:text="@string/setting_submit" />
      </LinearLayout>
```

在上述代码中，通过 Spinner 组件实现了一个供用户使用的下拉菜单。

2）编写处理文件 Setting.java，具体代码如下所示。

```
package com.iceskysl.map;

import com.iceskysl.map.R;

import android.app.Activity;
import android.content.Intent;
import android.content.SharedPreferences;
import android.os.Bundle;
import android.util.Log;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.widget.ArrayAdapter;
import android.widget.Button;
import android.widget.Spinner;
import android.widget.Toast;
```



```
public class Setting extends Activity {
    private static final String TAG = "Setting";
     //定义菜单需要的常量
     private static final int MENU_MAIN = Menu.FIRST + 1;
     private static final int MENU_NEW = MENU_MAIN + 1;
     private static final int MENU_BACK = MENU_NEW + 1;;

     // 保存个性化设置
     public static final String SETTING_INFOS = "SETTING_Infos";
     public static final String SETTING_GPS = "SETTING_Gps";
     public static final String SETTING_MAP = "SETTING_Map";
     public static final String SETTING_GPS_POSITON = "SETTING_Gps_p";
     public static final String SETTING_MAP_POSITON = "SETTING_Map_p";

     private Button button_setting_submit;
     private Spinner field_setting_gps;
     private Spinner field_setting_map_level;


    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.setting);
     setTitle(R.string.menu_setting);
            findViews();
            setListensers();
            restorePrefs();
    }

     private void findViews() {
            Log.d(TAG, "find Views");
            button_setting_submit = (Button) findViewById(R.id.setting_submit);
            field_setting_gps = (Spinner) findViewById(R.id.setting_gps);
          ArrayAdapter<CharSequence> adapter = ArrayAdapter.createFromResource(
                    this, R.array.gps, android.R.layout.simple_spinner_item);
adapter.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
          field_setting_gps.setAdapter(adapter);

            field_setting_map_level = (Spinner) findViewById(R.id.setting_map_level);
          ArrayAdapter<CharSequence> adapter2 = ArrayAdapter.createFromResource(
                    this, R.array.map, android.R.layout.simple_spinner_item);
adapter2.setDropDownViewResource(android.R.layout.simple_spinner_dropdown_item);
          field_setting_map_level.setAdapter(adapter2);
     }
```




```
        // Listen for button clicks
        private void setListensers() {
                Log.d(TAG, "set Listensers");
                button_setting_submit.setOnClickListener(setting_submit);
        }
        private Button.OnClickListener setting_submit = new Button.OnClickListener() {
                public void onClick(View v) {
                        Log.d(TAG, "onClick new_track..");
                        try {
                                String gps = (field_setting_gps.getSelectedItem().toString());
                                String map = (field_setting_map_level.getSelectedItem()
                                                .toString());
                                if (gps.equals("") || map.equals("")) {
                                        Toast.makeText(Setting.this,
                                                        getString(R.string.setting_null),
                                                        Toast.LENGTH_SHORT).show();
                                } else {
                                        //保存设定
                                        storePrefs();
                                        Toast.makeText(Setting.this,
                                                        getString(R.string.setting_ok),
                                                        Toast.LENGTH_SHORT).show();
                                        //跳转到主界面
                                        Intent intent = new Intent();
                                        intent.setClass(Setting.this, iTracks.class);
                                        startActivity(intent);
                                }
                        } catch (Exception err) {
                                Log.e(TAG, "error: " + err.toString());
                                Toast.makeText(Setting.this, getString(R.string.setting_fail),
                                                Toast.LENGTH_SHORT).show();
                        }
                }
        };

        // Restore preferences
        private void restorePrefs() {
                SharedPreferences settings = getSharedPreferences(SETTING_INFOS, 0);
                int setting_gps_p = settings.getInt(SETTING_GPS_POSITON, 0);
                int setting_map_level_p = settings.getInt(SETTING_MAP_POSITON, 0);
                Log.d(TAG, "restorePrefs: setting_gps= "+ setting_gps_p + ",setting_map_level=" +
setting_map_level_p);

                if (setting_gps_p != 0 && setting_map_level_p != 0) {
                        field_setting_gps.setSelection(setting_gps_p);
                        field_setting_map_level.setSelection(setting_map_level_p);
```



```
                button_setting_submit.requestFocus();
            }else if(setting_gps_p != 0 ){
                field_setting_gps.setSelection(setting_gps_p);
                field_setting_map_level.requestFocus();
            }else if(setting_map_level_p != 0){
                field_setting_map_level.setSelection(setting_map_level_p);
                field_setting_gps.requestFocus();
            }else{
                field_setting_gps.requestFocus();
            }
        }

        @Override
        protected void onStop(){
            super.onStop();
            Log.d(TAG, "save setting infos");
            // Save user preferences. We need an Editor object to
            // make changes. All objects are from android.context.Context
            storePrefs();
        }

        //保存个人设置
        private void storePrefs() {
            Log.d(TAG, "storePrefs setting infos");
            SharedPreferences settings = getSharedPreferences(SETTING_INFOS, 0);
            settings.edit()
                .putString(SETTING_GPS, field_setting_gps.getSelectedItem().toString())
                .putString(SETTING_MAP, field_setting_map_level.getSelectedItem().toString())
                .putInt(SETTING_GPS_POSITON, field_setting_gps.getSelectedItemPosition())
                .putInt(SETTING_MAP_POSITON, field_setting_map_level.getSelectedItemPosition())
                .commit();
        }

        // 初始化菜单
        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
            super.onCreateOptionsMenu(menu);
            menu.add(0, MENU_MAIN, 0, R.string.menu_main).setIcon(
                    R.drawable.icon).setAlphabeticShortcut('M');
            menu.add(0, MENU_NEW, 0, R.string.menu_new).setIcon(
                    R.drawable.new_track).setAlphabeticShortcut('N');
            menu.add(0, MENU_BACK, 0, R.string.menu_back).setIcon(
                    R.drawable.back).setAlphabeticShortcut('E');
            return true;
        }
```

```
        // 当一个菜单被选中的时候调用
        @Override
        public boolean onOptionsItemSelected(MenuItem item) {
            Intent intent = new Intent();
            switch (item.getItemId()) {
            case MENU_NEW:
                intent.setClass(Setting.this, NewTrack.class);
                startActivity(intent);
                return true;
            case MENU_MAIN:
                intent.setClass(Setting.this, iTracks.class);
                startActivity(intent);
                return true;
            case MENU_BACK:
               finish();
                break;
            }
            return true;
        }

    }
```

上述代码的具体实现流程如下。

第 1 步：声明需要的变量。

第 2 步：在 onCreat 中绑定 setting.xml 为其布局模板。

第 3 步：使用 setContentView()设定其对应的布局文件 setting.xml，使用 setTitle()设定其标题，进一步调用 findViews()查询到需要的操作组件，并调用 setListensers()给按钮设定单击监听器，最后调用 restorePrefs()将默认值或用户的历史选择值显示出来。

第 4 步：用 findViews()找到需要用到的组件。

其中下拉框中的内容是预先设置好的，定义在 array.xml 中，具体代码如下。

```
<?xml version="1.0" encoding="utf-8"?>
<resources>
    <!-- Used in View/setting.java -->
    <string-array name="gps">
        <item>1</item>
        <item>10</item>
        <item>20</item>
        <item>30</item>
        <item>40</item>
        <item>50</item>
    </string-array>
    <string-array name="map">
        <item>2</item>
```



```
                <item>3</item>
                <item>4</item>
                <item>5</item>
                <item>20</item>
                <item>30</item>
                <item>41</item>
                <item>52</item>
                <item>63</item>
                <item>74</item>
                <item>85</item>
                <item>96</item>
            </string-array>
        </resources>
```

至此，本模块的主要代码介绍完毕，执行后的效果如图 16−6 所示。



图 16−6　设置界面

16.2.5　帮助界面

当在图 16−4 中单击“帮助”按钮后，会弹出系统默认的帮助界面。本模块的具体实现流程如下。

1）编写布局文件 helps.xml，主要代码如下所示。

```
        <?xml version="1.0" encoding="utf-8"?>
        <LinearLayout xmlns:android="http://schemas.android.com/apk/res/android"
            android:orientation="vertical"
            android:layout_width="fill_parent"
            android:layout_height="fill_parent"
            >
```

```
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/version"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/version_text"
    />
<TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/helps_infos"
    />
   <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:autoLink="all"
    android:text="@string/helps_text"
 />
  <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:text="@string/author"
    />
    <TextView
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:autoLink="all"
    android:text="@string/author_text"
 />
</LinearLayout>
```

在上述代码中，通过 TextView 显示了各条帮助信息。

2）编写处理文件 helps.java，主要代码如下所示。

```
package com.iceskysl.map;

import com.iceskysl.map.R;

import android.app.Activity;
import android.content.Intent;
import android.os.Bundle;
import android.view.Menu;
import android.view.MenuItem;
```



```
public class Helps extends Activity {
    //定义菜单需要的常量
    private static final int MENU_MAIN = Menu.FIRST + 1;
    private static final int MENU_NEW = MENU_MAIN + 1;
    private static final int MENU_BACK = MENU_NEW + 1;;
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.helps);
    setTitle(R.string.menu_helps);
    }
    // 初始化菜单
    @Override
    public boolean onCreateOptionsMenu(Menu menu) {
        super.onCreateOptionsMenu(menu);
        menu.add(0, MENU_MAIN, 0, R.string.menu_main).setIcon(
                R.drawable.icon).setAlphabeticShortcut('M');
        menu.add(0, MENU_NEW, 0, R.string.menu_new).setIcon(
                R.drawable.new_track).setAlphabeticShortcut('N');
        menu.add(0, MENU_BACK, 0, R.string.menu_back).setIcon(
                R.drawable.back).setAlphabeticShortcut('E');
        return true;
    }

    // 当一个菜单被选中的时候调用
    @Override
    public boolean onOptionsItemSelected(MenuItem item) {
        Intent intent = new Intent();
        switch (item.getItemId()) {
        case MENU_NEW:
            intent.setClass(Helps.this, NewTrack.class);
            startActivity(intent);
            return true;
        case MENU_MAIN:
            intent.setClass(Helps.this, iTracks.class);
            startActivity(intent);
            return true;
        case MENU_BACK:
            finish();
            break;
        }
        return true;
    }
}
```

在上述代码中，首先在 onCreate 方法中设定其对应的布局文件为 helps.xml，然后添加菜

单和菜单对应的功能。

至此，本模块的主要代码介绍完毕，执行后的效果如图 16-7 所示。



图 16-7　帮助界面

16.2.6　地图界面

前面介绍的都是主菜单中的选项，下面开始讲解比较复杂的功能：将地图在 Android 手机中显示。前面已经讲解了 com.google.android.maps 的基本知识，通过其中的 MapView 即可方便地实现编程工作。基本实现流程如下所示。

1．申请APIKey

1）打开 Eclipse，依次单击“Windows”→“Preferences”→“Android”→“Build”，查看默认的 debug keystore 位置，如图 16-8 所示。

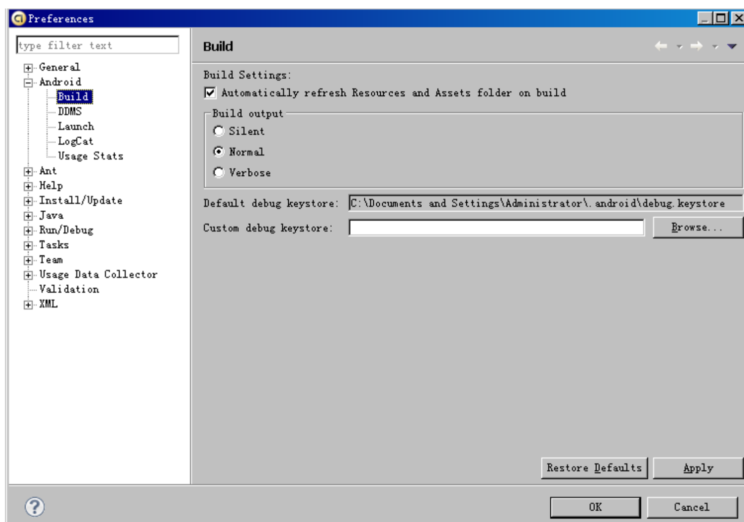


图 16-8　debug keystore 位置



2）打开运行，输入“cmd”后按“确定”按钮，在命令行中执行如下命令。

```
keytool -list -alias androiddebugkey -keystore "C:\Documents and Settings\Administrator\.android\debug.keystore" -storepass android -keypass android
```

通过上述命令获取 MD5 指纹，此时系统会提示输入 keystore 代码，输入 android 后会显示要获取的指纹。

3）打开网址 http://code.google.com/intl/zh-CN/android/maps-api-signup.html，如图 16-9 所示。输入得到的 MD5 指纹，然后单击 Generate API Key 申请获取 API Key。

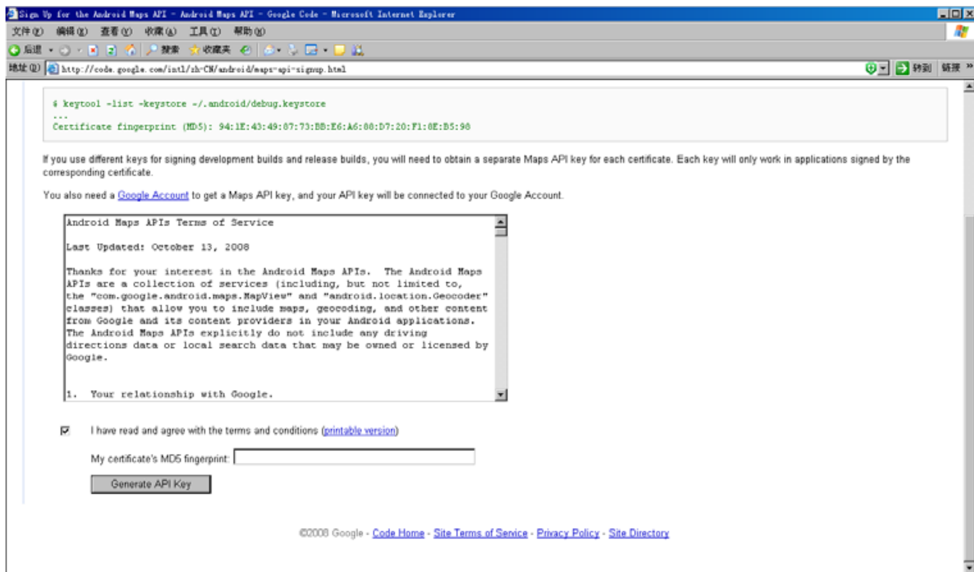


图 16-9　获取 APIKey

2. 编码

下面开始介绍编码过程，具体流程如下。

1）编写布局文件 show_track.xml，具体代码如下所示。

```
<?xml version="1.0" encoding="utf-8"?>
<FrameLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    >
<view android:id="@+id/mv"
    class="com.google.android.maps.MapView"
    android:layout_width="fill_parent"
    android:layout_height="fill_parent"
    android:layout_weight="1"
    android:apiKey="01Yu9W3X3vbr4UFCa_OlwALuXpyD__ocgLaiYNw"
    />
```




```
<LinearLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:orientation="horizontal"
    android:layout_width="fill_parent"
    android:layout_height="wrap_content"
    android:background="#550000ff"
    android:padding="1px"
    >
<Button android:id="@+id/sat"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginLeft="40px"
    android:text="@string/satellite" />
<Button android:id="@+id/traffic"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/traffic" />
<Button android:id="@+id/streetview"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="@string/street" />

<Button android:id="@+id/gps"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="GPS" />
</LinearLayout>
<LinearLayout xmlns:android=
    "http://schemas.android.com/apk/res/android"
    android:orientation="vertical"
    android:layout_width="wrap_content"
    android:layout_height="fill_parent"
    android:background="#550000ff"
    android:padding="1px"
    >
<Button android:id="@+id/zin"
    android:layout_width="wrap_content"
    android:layout_height="wrap_content"
    android:layout_marginTop="30px"
    android:text="+" />
<Button android:id="@+id/zout"
      android:layout_width="wrap_content"
      android:layout_height="wrap_content"
      android:text="-" />
<Button android:id="@+id/pann"
      android:layout_width="wrap_content"
```



```
        android:layout_height="wrap_content"
        android:text="N" />
    <Button android:id="@+id/pane"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="E" />

    <Button android:id="@+id/panw"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="W" />
    <Button android:id="@+id/pans"
        android:layout_width="wrap_content"
        android:layout_height="wrap_content"
        android:text="S" />
    </LinearLayout>
    </FrameLayout>
```

通过上述代码，通过 MapView 组件来显示地图，并通过设置的按钮来控制地图，例如，放大、缩小、移动和模式的转换。

2）编写处理文件 ShowTrack.java，具体代码如下所示。

```
package com.iceskysl.map;

import android.content.Context;
import android.content.Intent;
import android.content.SharedPreferences;
import android.content.res.Resources;
import android.database.Cursor;
import android.graphics.Canvas;
import android.graphics.Paint;
import android.graphics.Point;
import android.graphics.RectF;
import android.graphics.Paint.Style;
import android.location.Location;
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.util.Log;
import android.view.KeyEvent;
import android.view.Menu;
import android.view.MenuItem;
import android.view.View;
import android.view.View.OnClickListener;
import android.widget.Button;
import android.widget.Toast;
```

```
import com.google.android.maps.GeoPoint;
import com.google.android.maps.MapActivity;
import com.google.android.maps.MapController;
import com.google.android.maps.MapView;
import com.google.android.maps.MyLocationOverlay;
import com.google.android.maps.Overlay;
import com.iceskysl.map.R;

public class ShowTrack extends MapActivity {
    // 定义菜单需要的常量
    private static final int MENU_NEW = Menu.FIRST + 1;
    private static final int MENU_CON = MENU_NEW + 1;
    private static final int MENU_DEL = MENU_CON + 1;
    private static final int MENU_MAIN = MENU_DEL + 1;

    private TrackDbAdapter mDbHelper;
    private LocateDbAdapter mlcDbHelper;

    private static final String TAG = "ShowTrack";
    private static MapView mMapView;
    private MapController mc;

    protected MyLocationOverlay mOverlayController;
    private Button mZin;
    private Button mZout;
    private Button mPanN;
    private Button mPanE;
    private Button mPanW;
    private Button mPanS;
    private Button mGps;
    private Button mSat;
    private Button mTraffic;
    private Button mStreetview;
    private String mDefCaption = "";
    private GeoPoint mDefPoint;

    private LocationManager lm;
    private LocationListener locationListener;

    private int track_id;
    private Long rowId;

    /** Called when the activity is first created. */
    public void onCreate(Bundle icicle) {
        super.onCreate(icicle);
```



```
        setContentView(R.layout.show_track);
        findViews();
        centerOnGPSPosition();
        revArgs();
        // //////////////
          //mDbHelper = new TrackDbAdapter(this);
          //mDbHelper.open();

        //mlcDbHelper = Track.getDbHelp();
                //new LocateDbAdapter(this);
        //mlcDbHelper.open();

        paintLocates();
        startTrackService();
}

private void startTrackService() {
        Intent i = new Intent("com.iceskysl.iTracks.START_TRACK_SERVICE");
        i.putExtra(LocateDbAdapter.TRACKID, track_id);
      startService(i);
}

private void stopTrackService() {
      stopService(new Intent("com.iceskysl.iTracks.START_TRACK_SERVICE"));
}

private void paintLocates() {
        mlcDbHelper = new LocateDbAdapter(this);
        mlcDbHelper.open();
        Cursor mLocatesCursor = mlcDbHelper.getTrackAllLocates(track_id);
        startManagingCursor(mLocatesCursor);
        Resources resources = getResources();
        Overlay overlays = new LocateOverLay(resources
                        .getDrawable(R.drawable.icon), mLocatesCursor);
        mMapView.getOverlays().add(overlays);
        mlcDbHelper.close();
}

private void revArgs() {
        Log.d(TAG, "revArgs.");
        Bundle extras = getIntent().getExtras();
        if (extras != null) {
                String name = extras.getString(TrackDbAdapter.NAME);
                //String desc = extras.getString(TrackDbAdapter.DESC);
                rowId = extras.getLong(TrackDbAdapter.KEY_ROWID);
                track_id = rowId.intValue();
```




```
                Log.d(TAG, "rowId=" + rowId);
                if (name != null) {
                    setTitle(name);
                }
            }
        }

        protected boolean isRouteDisplayed() {
            // TODO Auto-generated method stub
            return false;
        }

        private void findViews() {
            Log.d(TAG, "find Views");
            // Get the map view from resource file
            mMapView = (MapView) findViewById(R.id.mv);
            mc = mMapView.getController();

            SharedPreferences settings = getSharedPreferences(Setting.SETTING_INFOS, 0);
            String setting_gps = settings.getString(Setting.SETTING_MAP, "10");
            mc.setZoom(Integer.parseInt(setting_gps));

            // Set up the button for "Pan East"
            mPanE = (Button) findViewById(R.id.sat);
            mPanE.setOnClickListener(new OnClickListener() {
                // @Override
                public void onClick(View arg0) {
                    panEast();
                }
            });
            // Set up the button for "Zoom In"
            mZin = (Button) findViewById(R.id.zin);
            mZin.setOnClickListener(new OnClickListener() {
                // @Override
                public void onClick(View arg0) {
                    zoomIn();
                }
            });
            // Set up the button for "Zoom Out"
            mZout = (Button) findViewById(R.id.zout);
            mZout.setOnClickListener(new OnClickListener() {
                // @Override
                public void onClick(View arg0) {
                    zoomOut();
                }
            });
```



```
// Set up the button for "Pan North"
mPanN = (Button) findViewById(R.id.pann);
mPanN.setOnClickListener(new OnClickListener() {
        // @Override
        public void onClick(View arg0) {
                panNorth();
        }
});

// Set up the button for "Pan East"
mPanE = (Button) findViewById(R.id.pane);
mPanE.setOnClickListener(new OnClickListener() {
        // @Override
        public void onClick(View arg0) {
                panEast();
        }
});

// Set up the button for "Pan West"
mPanW = (Button) findViewById(R.id.panw);
mPanW.setOnClickListener(new OnClickListener() {
        // @Override
        public void onClick(View arg0) {
                panWest();
        }
});
// Set up the button for "Pan South"
mPanS = (Button) findViewById(R.id.pans);
mPanS.setOnClickListener(new OnClickListener() {
        // @Override
        public void onClick(View arg0) {
                panSouth();
        }
});

// Set up the button for "GPS"
mGps = (Button) findViewById(R.id.gps);
mGps.setOnClickListener(new OnClickListener() {
        // @Override
        public void onClick(View arg0) {
                centerOnGPSPosition();
        }
});
// Set up the button for "Satellite toggle"
mSat = (Button) findViewById(R.id.sat);
mSat.setOnClickListener(new OnClickListener() {
```

```
                // @Override
                public void onClick(View arg0) {
                        toggleSatellite();
                }
        });

        // Set up the button for "Traffic toggle"
        mTraffic = (Button) findViewById(R.id.traffic);
        mTraffic.setOnClickListener(new OnClickListener() {
                // @Override
                public void onClick(View arg0) {
                        toggleTraffic();
                }
        });

        // Set up the button for "Traffic toggle"
        mStreetview = (Button) findViewById(R.id.streetview);
        mStreetview.setOnClickListener(new OnClickListener() {
                // @Override
                public void onClick(View arg0) {
                        toggleStreetView();
                }
        });

        // ---use the LocationManager class to obtain GPS locations---
        lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
        locationListener = new MyLocationListener();
        lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
                        locationListener);
}

public boolean onKeyDown(int keyCode, KeyEvent event) {
        Log.d(TAG, "onKeyDown");
        if (keyCode == KeyEvent.KEYCODE_DPAD_LEFT) {
                panWest();
                return true;
        } else if (keyCode == KeyEvent.KEYCODE_DPAD_RIGHT) {
                panEast();
                return true;
        } else if (keyCode == KeyEvent.KEYCODE_DPAD_UP) {
                panNorth();
                return true;
        } else if (keyCode == KeyEvent.KEYCODE_DPAD_DOWN) {
                panSouth();
                return true;
        }
```



```
        return false;
    }

    public void panWest() {
        GeoPoint pt = new GeoPoint(mMapView.getMapCenter().getLatitudeE6(),
                mMapView.getMapCenter().getLongitudeE6()
                        - mMapView.getLongitudeSpan() / 4);
        mc.setCenter(pt);
    }

    public void panEast() {
        GeoPoint pt = new GeoPoint(mMapView.getMapCenter().getLatitudeE6(),
                mMapView.getMapCenter().getLongitudeE6()
                        + mMapView.getLongitudeSpan() / 4);
        mc.setCenter(pt);
    }

    public void panNorth() {
        GeoPoint pt = new GeoPoint(mMapView.getMapCenter().getLatitudeE6()
                + mMapView.getLatitudeSpan() / 4, mMapView.getMapCenter()
                .getLongitudeE6());
        mc.setCenter(pt);
    }

    public void panSouth() {
        GeoPoint pt = new GeoPoint(mMapView.getMapCenter().getLatitudeE6()
                - mMapView.getLatitudeSpan() / 4, mMapView.getMapCenter()
                .getLongitudeE6());
        mc.setCenter(pt);
    }

    public void zoomIn() {
        mc.zoomIn();
    }

    public void zoomOut() {
        mc.zoomOut();
    }

    public void toggleSatellite() {
        mMapView.setSatellite(true);
        mMapView.setStreetView(false);
        mMapView.setTraffic(false);
    }

    public void toggleTraffic() {
```

```
        mMapView.setTraffic(true);
        mMapView.setSatellite(false);
        mMapView.setStreetView(false);
    }

    public void toggleStreetView() {
        mMapView.setStreetView(true);
        mMapView.setSatellite(false);
        mMapView.setTraffic(false);
    }

    private void centerOnGPSPosition() {
        Log.d(TAG, "centerOnGPSPosition");
        String provider = "gps";
        LocationManager lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);

        Location loc = lm.getLastKnownLocation(provider);
        loc = lm.getLastKnownLocation(provider);

        mDefPoint = new GeoPoint((int) (loc.getLatitude() * 1000000),
                (int) (loc.getLongitude() * 1000000));
        mDefCaption = "I'm Here.";
        mc.animateTo(mDefPoint);
        mc.setCenter(mDefPoint);
        // show Overlay on map.
        MyOverlay mo = new MyOverlay();
        mo.onTap(mDefPoint, mMapView);
        mMapView.getOverlays().add(mo);
    }

    // This is used draw an overlay on the map
    protected class MyOverlay extends Overlay {
        @Override
        public void draw(Canvas canvas, MapView mv, boolean shadow) {
            Log.d(TAG, "MyOverlay::darw..mDefCaption=" + mDefCaption);
            super.draw(canvas, mv, shadow);

            if (mDefCaption.length() == 0) {
                return;
            }

            Paint p = new Paint();
            int[] scoords = new int[2];
            int sz = 5;
```



```
            // Convert to screen coords
            Point myScreenCoords = new Point();
            mMapView.getProjection().toPixels(mDefPoint, myScreenCoords);
            // mMapView.set
            // mv.getPointXY(mDefPoint, scoords);
            // Draw point caption and its bounding rectangle
            scoords[0] = myScreenCoords.x;
            scoords[1] = myScreenCoords.y;
            p.setTextSize(14);
            p.setAntiAlias(true);

            int sw = (int) (p.measureText(mDefCaption) + 0.5f);
            int sh = 25;
            int sx = scoords[0] - sw / 2 - 5;
            int sy = scoords[1] - sh - sz - 2;
            RectF rec = new RectF(sx, sy, sx + sw + 10, sy + sh);

            p.setStyle(Style.FILL);
            p.setARGB(128, 255, 0, 0);

            canvas.drawRoundRect(rec, 5, 5, p);

            p.setStyle(Style.STROKE);
            p.setARGB(255, 255, 255, 255);
            canvas.drawRoundRect(rec, 5, 5, p);
            // canvas.d

            canvas.drawText(mDefCaption, sx + 5, sy + sh - 8, p);

            // Draw point body and outer ring
            p.setStyle(Style.FILL);
            p.setARGB(88, 255, 0, 0);
            p.setStrokeWidth(1);
            RectF spot = new RectF(scoords[0] - sz, scoords[1] + sz, scoords[0]
                    + sz, scoords[1] - sz);
            canvas.drawOval(spot, p);

            p.setARGB(255, 255, 0, 0);
            p.setStyle(Style.STROKE);
            canvas.drawCircle(scoords[0], scoords[1], sz, p);
        }
    }

    //
    protected class MyLocationListener implements LocationListener {
```

```
			@Override
			public void onLocationChanged(Location loc) {
				Log.d(TAG, "MyLocationListener::onLocationChanged..");
				if (loc != null) {
					Toast.makeText(
							getBaseContext(),
							"Location changed : Lat: " + loc.getLatitude()
									+ " Lng: " + loc.getLongitude(),
							Toast.LENGTH_SHORT).show();
					// Set up the overlay controller
					// mOverlayController = mMapView.createOverlayController();
					mDefPoint = new GeoPoint((int) (loc.getLatitude() * 1000000),
							(int) (loc.getLongitude() * 1000000));
					mc.animateTo(mDefPoint);
					mc.setCenter(mDefPoint);
					// show on the map.
					mDefCaption = "Lat: " + loc.getLatitude() + ",Lng: "
							+ loc.getLongitude();
					MyOverlay mo = new MyOverlay();
					mo.onTap(mDefPoint, mMapView);
					mMapView.getOverlays().add(mo);
					// //////////
					//if(mlcDbHelper == null){
					//	mlcDbHelper.open();
					//}
					//mlcDbHelper.createLocate(track_id,		loc.getLongitude(),loc.getLatitude(),
loc.getAltitude());
				}
			}

			@Override
			public void onProviderDisabled(String provider) {
				Toast.makeText(
						getBaseContext(),
						"ProviderDisabled.",
						Toast.LENGTH_SHORT).show();		}

			@Override
			public void onProviderEnabled(String provider) {
				Toast.makeText(
						getBaseContext(),
						"ProviderEnabled,provider:"+provider,
						Toast.LENGTH_SHORT).show();		}

			@Override
			public void onStatusChanged(String provider, int status, Bundle extras) {
```



```
                // TODO Auto-generated method stub
            }
        }

        // 初始化菜单
        @Override
        public boolean onCreateOptionsMenu(Menu menu) {
            super.onCreateOptionsMenu(menu);
            menu.add(0, MENU_CON, 0, R.string.menu_con).setIcon(
                    R.drawable.con_track).setAlphabeticShortcut('C');
            menu.add(0, MENU_DEL, 0, R.string.menu_del).setIcon(R.drawable.delete)
                    .setAlphabeticShortcut('D');
            menu.add(0, MENU_NEW, 0, R.string.menu_new).setIcon(
                    R.drawable.new_track).setAlphabeticShortcut('N');
            menu.add(0, MENU_MAIN, 0, R.string.menu_main).setIcon(R.drawable.icon)
                    .setAlphabeticShortcut('M');
            return true;
        }

        // 当一个菜单被选中的时候调用
        @Override
        public boolean onOptionsItemSelected(MenuItem item) {
            Intent intent = new Intent();
            switch (item.getItemId()) {
            case MENU_NEW:
                intent.setClass(ShowTrack.this, NewTrack.class);
                startActivity(intent);
                return true;
            case MENU_CON:
                // TODO: 继续跟踪选择的记录
                startTrackService();
                return true;
            case MENU_DEL:
                 mDbHelper = new TrackDbAdapter(this);
                 mDbHelper.open();
                if (mDbHelper.deleteTrack(rowId)) {
                    mDbHelper.close();
                    intent.setClass(ShowTrack.this, iTracks.class);
                    startActivity(intent);
                }else{
                    mDbHelper.close();
                }
                return true;
            case MENU_MAIN:
                intent.setClass(ShowTrack.this, iTracks.class);
                startActivity(intent);
```

```
                    break;
                }
                return true;
            }

            @Override
            protected void onStop() {
                super.onStop();
                Log.d(TAG, "onStop");
                // mDbHelper.close();
                //mlcDbHelper.close();
            }

            @Override
           public void onDestroy() {
                Log.d(TAG, "onDestroy.");
              super.onDestroy();
              stopTrackService();
              }
        }
```

上述代码的具体实现流程如下。

第 1 步：通过 findViews 来确定要使用的控件，并绑定需要响应的事件。

第 2 步：通过 findViews 实现对地图的处理，首先获取布局中的 MapView，使用 getController 得到一个 MapController，然后注册一个基于 locationListener 的 MyLocationListener。

第 3 步：实现处理按钮的具体处理方法的代码，具体原理比较简单，即首先获取地图中心，然后向 4 个方向移动 1/4 距离。

第 4 步：单击“GPS”按钮的响应方法 centerOnGPSPosition，即将地图定位到当前 GPS 指定的位置。

第 5 步：通过 Overylay 抽象类重载实现其 draw 方法。

至此，本模块主要代码介绍完毕，执行后的效果如图 16-10 所示。



图 16-10　地图展示界面



16.2.7 数据存取

在前面介绍的系统需求分析中，系统要求将每次跟踪的目标位置保存在数据库中，并且每次改变后都要保存起来。本项目的个性化配置信息保存在 SharedPreferences 中，在此将需要存取的数据放在数据库中。在 Android 中，存取数据库的方法有两种，一种通过 help 类继承 SQLiteDatabase 相关类绑定 SQL；另外一种是使用 ContentProvideer 进行封装。

1．创建数据库

本项目需要同时操作数据库中的两个表，在此先在文件 DbAdapter.java 中创建一个名为 DbAdapter 的类，具体实现代码如下所示。

```
package com.iceskysl.map;

import android.content.Context;
import android.database.sqlite.SQLiteDatabase;
import android.database.sqlite.SQLiteOpenHelper;
import android.util.Log;

public class DbAdapter {
    private static final String TAG = "DbAdapter";
    private static final String DATABASE_NAME = "iTracks.db";
    private static final int DATABASE_VERSION = 1;

    public class DatabaseHelper extends SQLiteOpenHelper {
        public DatabaseHelper(Context context) {
            super(context, DATABASE_NAME, null, DATABASE_VERSION);
        }
        @Override
        public void onCreate(SQLiteDatabase db) {
            String tracks_sql = "CREATE TABLE " + TrackDbAdapter.TABLE_NAME + " ("
            + TrackDbAdapter.ID   + " INTEGER primary key autoincrement, "
            + TrackDbAdapter.NAME     + " text not null, "
            + TrackDbAdapter.DESC + " text ,"
            + TrackDbAdapter.DIST + " LONG ,"
            + TrackDbAdapter.TRACKEDTIME + " LONG ,"
            + TrackDbAdapter.LOCATE_COUNT + " INTEGER, "
            + TrackDbAdapter.CREATED + " text, "
            + TrackDbAdapter.AVGSPEED + " LONG, "
            + TrackDbAdapter.MAXSPEED + " LONG ,"
            + TrackDbAdapter.UPDATED + " text "
            + ");";
            Log.i(TAG, tracks_sql);
            db.execSQL(tracks_sql);

            String locats_sql = "CREATE TABLE " + LocateDbAdapter.TABLE_NAME + " ("
            + LocateDbAdapter.ID + " INTEGER primary key autoincrement, "
```

```
                  + LocateDbAdapter.TRACKID      + " INTEGER not null, "
                  + LocateDbAdapter.LON + " DOUBLE ,"
                  + LocateDbAdapter.LAT + " DOUBLE ,"
                  + LocateDbAdapter.ALT + " DOUBLE ,"
                  + LocateDbAdapter.CREATED + " text "
                  + ");";
                  Log.i(TAG, locats_sql);
                  db.execSQL(locats_sql);
              }
              @Override
              public void onUpgrade(SQLiteDatabase db, int oldVersion, int newVersion) {
                  db.execSQL("DROP TABLE IF EXISTS " + LocateDbAdapter.TABLE_NAME +
";");

                  db.execSQL("DROP TABLE IF EXISTS " + TrackDbAdapter.TABLE_NAME + ";");
                  onCreate(db);
              }
          }

      }
```

在上述代码中，重新定义了 SQLiteOpenHelper 的 onCreate 方法和 onUpgrade 方法，通过这两个方法实现了创建和升级数据库的脚本。

2．数据库操作

本功能实现了对两个表操作的封装处理，因为共用了同一个数据库，所以只需从前面创建的 bAdapter 中继续继承即可，在此继承出了两个类：TrackDbAdapter 和 LocateDbAdapter。通过对这两个类的封装，实现了对数据表操作。

1）TrackDbAdapter 类是在文件 TrackDbAdapter.java 中定义的，具体代码如下所示。

```
package com.iceskysl.map;

import java.util.Calendar;

import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.util.Log;

public class TrackDbAdapter extends DbAdapter{
    private static final String TAG = "TrackDbAdapter";

    public static final String TABLE_NAME = "tracks";

    public static final String ID = "_id";
    public static final String KEY_ROWID = "_id";
```



```
public static final String NAME = "name";
public static final String DESC = "desc";
public static final String DIST = "distance";
public static final String TRACKEDTIME = "tracked_time";
public static final String LOCATE_COUNT = "locats_count";
public static final String CREATED = "created_at";
public static final String UPDATED = "updated_at";
public static final String AVGSPEED = "avg_speed";
public static final String MAXSPEED = "max_speed";

private DatabaseHelper mDbHelper;
private SQLiteDatabase mDb;
private final Context mCtx;

public TrackDbAdapter(Context ctx) {
        this.mCtx = ctx;
}

public TrackDbAdapter open() throws SQLException {
        mDbHelper = new DatabaseHelper(mCtx);
        mDb = mDbHelper.getWritableDatabase();
        return this;
}

public void close() {
        mDbHelper.close();
}

public Cursor getTrack(long rowId) throws SQLException {
        Cursor mCursor =
        mDb.query(true, TABLE_NAME, new String[] { KEY_ROWID, NAME,
                        DESC, CREATED }, KEY_ROWID + "=" + rowId, null, null,
                        null, null, null);
        if (mCursor != null) {
                mCursor.moveToFirst();
        }
        return mCursor;
}

public long createTrack(String name, String desc) {
        Log.d(TAG, "createTrack.");
        ContentValues initialValues = new ContentValues();
        initialValues.put(NAME, name);
        initialValues.put(DESC, desc);
        Calendar calendar = Calendar.getInstance();
        String created = calendar.get(Calendar.YEAR) + "-" +calendar.get(Calendar.MONTH) + "-"
```

```
+    calendar.get(Calendar.DAY_OF_MONTH) + " "
                                        + calendar.get(Calendar.HOUR_OF_DAY) + ":"
                                        + calendar.get(Calendar.MINUTE) + ":"    + calendar.get(Calendar.SECOND);
                        initialValues.put(CREATED, created);
                        initialValues.put(UPDATED, created);
                        return mDb.insert(TABLE_NAME, null, initialValues);
                }

                //
                public boolean deleteTrack(long rowId) {
                        return mDb.delete(TABLE_NAME, KEY_ROWID + "=" + rowId, null) > 0;
                }

                public Cursor getAllTracks() {
                        return mDb.query(TABLE_NAME, new String[] { ID, NAME,
                                        DESC, CREATED }, null, null, null, null, "updated_at desc");
                }

                public boolean updateTrack(long rowId, String name, String desc) {
                        ContentValues args = new ContentValues();
                        args.put(NAME, name);
                        args.put(DESC, desc);
                        Calendar calendar = Calendar.getInstance();
                        String updated = calendar.get(Calendar.YEAR) + "-" +calendar.get(Calendar.MONTH) + "-"
+    calendar.get(Calendar.DAY_OF_MONTH) + " "
                        + calendar.get(Calendar.HOUR_OF_DAY) + ":"
                        + calendar.get(Calendar.MINUTE) + ":"    + calendar.get(Calendar.SECOND);
                        args.put(UPDATED, updated);
                        return mDb.update(TABLE_NAME, args, KEY_ROWID + "=" + rowId, null) > 0;
                }

        }
```

在上述代码中，首先声明了一些常量，然后根据需要的操作功能定义了具体方法。

2）LocateDbAdapter 类是在文件 LocateDbAdapter.java 中实现的，具体代码如下所示。

```
package com.iceskysl.map;

import java.util.Calendar;
import android.content.ContentValues;
import android.content.Context;
import android.database.Cursor;
import android.database.SQLException;
import android.database.sqlite.SQLiteDatabase;
import android.util.Log;

public class LocateDbAdapter extends DbAdapter {
```



```
private static final String TAG = "LocateDbAdapter";
public static final String TABLE_NAME = "locates";

public static final String ID = "_id";
public static final String TRACKID = "track_id";
public static final String LON = "longitude";
public static final String LAT = "latitude";
public static final String ALT = "altitude";
public static final String CREATED = "created_at";

private DatabaseHelper mDbHelper;
private SQLiteDatabase mDb;
private final Context mCtx;

public LocateDbAdapter(Context ctx) {
    this.mCtx = ctx;
}

public LocateDbAdapter open() throws SQLException {
    mDbHelper = new DatabaseHelper(mCtx);
    mDb = mDbHelper.getWritableDatabase();
    return this;
}

public void close() {
    mDbHelper.close();
}

public Cursor getLocate(long rowId) throws SQLException {
    Cursor mCursor =
    mDb.query(true, TABLE_NAME, new String[] { ID, LON,
            LAT, ALT, CREATED }, ID + "=" + rowId, null, null,
            null, null, null);
    if (mCursor != null) {
        mCursor.moveToFirst();
    }
    return mCursor;
}

public long createLocate(int track_id, Double longitude, Double latitude ,Double altitude) {
    Log.d(TAG, "createLocate.");
    ContentValues initialValues = new ContentValues();
    initialValues.put(TRACKID, track_id);
    initialValues.put(LON, longitude);
    initialValues.put(LAT, latitude);
    initialValues.put(ALT, altitude);
```

```
                Calendar calendar = Calendar.getInstance();
                String created = calendar.get(Calendar.YEAR) + "-" +calendar.get(Calendar.MONTH) + "-"
+   calendar.get(Calendar.DAY_OF_MONTH) + " "
                        + calendar.get(Calendar.HOUR_OF_DAY) + ":"
                        + calendar.get(Calendar.MINUTE) + ":"   + calendar.get(Calendar.SECOND);
                initialValues.put(CREATED, created);
                return mDb.insert(TABLE_NAME, null, initialValues);
        }

        //
        public boolean deleteLocate(long rowId) {
                return mDb.delete(TABLE_NAME, ID + "=" + rowId, null) > 0;
        }



        public Cursor getTrackAllLocates(int trackId) {
                return mDb.query(TABLE_NAME, new String[] { ID,TRACKID, LON,
                        LAT, ALT,CREATED }, "track_id=?", new String[] {String.valueOf(trackId)},
null, null, "created_at asc");
        }
    }
```

在上述代码中，也是首先声明了一些常量，然后根据需要的操作功能定义了具体方法。

16.2.8　实现Service服务

项目的基本要求是，切换一个界面不会影响到对目标的追踪。即即使来到另外一个界面，程序也需要在后台进行跟踪和记录。所以需要用到 Service 服务，首先在文件 AndroidManifest.xml 中加入对 Service 的声明，代码如下。

```
<service android:name=".Track">
        <intent-filter>
                <action android:name="com.iceskysl.map.START_TRACK_SERVICE" />
                <category android:name="android.intent.category.default" />
        </intent-filter>
</service>
```

通过上述代码添加了一个名为 Track 的 Service，并设定了其名字为“com.iceskysl.map. START_TRACK_SERVICE”。接着看处理文件 Track.java 的具体实现代码。

```
package com.iceskysl.map;

import android.app.Service;
import android.content.Context;
import android.content.Intent;
import android.location.Location;
```



```
import android.location.LocationListener;
import android.location.LocationManager;
import android.os.Bundle;
import android.os.IBinder;
import android.util.Log;
import android.widget.Toast;

public class Track extends Service {
    private static final String TAG = "Track";

    private LocationManager lm;
    private LocationListener locationListener;

    static LocateDbAdapter mlcDbHelper = null;
    private int track_id;

    @Override
    public IBinder onBind(Intent arg0) {
        Log.d(TAG, "onBind.");
        return null;
    }

    public void onStart(Intent intent, int startId) {
        Log.d(TAG, "onStart.");
      super.onStart(intent, startId);
      startDb();
        Bundle extras = intent.getExtras();
        if (extras != null) {
            track_id = extras.getInt(LocateDbAdapter.TRACKID);
        }
        Log.d(TAG, "track_id =" + track_id);
        // ---use the LocationManager class to obtain GPS locations---
        lm = (LocationManager) getSystemService(Context.LOCATION_SERVICE);
        locationListener = new MyLocationListener();
        lm.requestLocationUpdates(LocationManager.GPS_PROVIDER, 0, 0,
                locationListener);
    }

    private void startDb() {
        if(mlcDbHelper == null){
            mlcDbHelper = new LocateDbAdapter(this);
            mlcDbHelper.open();
        }
    }

    private void stopDb() {
```

```
            if(mlcDbHelper != null){
                mlcDbHelper.close();
            }
        }


        public static LocateDbAdapter getDbHelp(){
            return mlcDbHelper;
        }


        public void onDestroy() {
            Log.d(TAG, "onDestroy.");
            super.onDestroy();
            stopDb();
        }
        protected class MyLocationListener implements LocationListener {

            @Override
            public void onLocationChanged(Location loc) {
                Log.d(TAG, "MyLocationListener::onLocationChanged..");
                if (loc != null) {
                    // //////////
                    if(mlcDbHelper == null){
                        mlcDbHelper.open();
                    }
                    mlcDbHelper.createLocate(track_id,        loc.getLongitude(),loc.getLatitude(),
loc.getAltitude());
                }
            }

            @Override
            public void onProviderDisabled(String provider) {
                Toast.makeText(
                        getBaseContext(),
                        "ProviderDisabled.",
                        Toast.LENGTH_SHORT).show();        }

            @Override
            public void onProviderEnabled(String provider) {
                Toast.makeText(
                        getBaseContext(),
                        "ProviderEnabled,provider:"+provider,
                        Toast.LENGTH_SHORT).show();        }
            @Override
            public void onStatusChanged(String provider, int status, Bundle extras) {
```



```
                    // TODO Auto-generated method stub
                }
            }
        }
```

在上述代码中，Track 继承了 Service 类，然后在 onStart 中连接了数据库，接收了参数并设定了监听器，并使用了 MyLocationListener，当在位置变化（onLoacationChanged）的时候，调用前面数据存储部分已经实现的 mlcDbHelper.createLocate 方法，将位置信息和接收到的参数写入到数据库中。

ISBN 978-7-111-34928-0
策划编辑：丁 诚
封面设计：子时文化 ZiShi Culture

内容简介

本书循序渐进的介绍了Android技术的基础知识，并通过实例教学的方式讲解了Android技术在各个领域的具体应用过程。全书分为16章，其中第1～5章是基础篇，讲解了Android的发展前景和开发环境的搭建过程；第6～13章是核心技术篇，详细讲解了Android技术的核心知识，并对程序优化进行了详细剖析；第14～16章是综合实战应用篇，通过3个综合实例详细讲解了Android技术常用的开发流程。

本书定位于Android的初、中级用户，可作为初学者的自学手册，也可以作为有一定基础的程序员的参考书。

上架建议 计算机/移动与嵌入式开发

地址：北京市百万庄大街22号
电话服务
社服中心：(010)88361066
销售一部：(010)68326294
销售二部：(010)88379649
读者购书热线：(010)88379203
邮政编码：100037
网络服务
门户网：http:www.cmpbook.com
教材网：http:www.cmpedu.com
封面无防伪标均为盗版

ISBN 978-7-111-34928-0


定价：69.80元（含1DVD）